Instituto Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Biblioteca para la implementación de algoritmos de Control Automático en la plataforma Hawkboard (OMAPL138)

Informe de Proyecto de Graduación para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura

Manuel Leiva Fernández

Cartago, Noviembre del 2010

ESCUELA DE INGENIERIA ELECTRÓNICA
PROYECTO DE GRADUACIÓN

TRIBUNAL EVALUADOR

Proyecto de Graduación defendido ante el presente Tribunal Evaluador como requisito para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura, del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal

Ing. Pablo Alvarado Moya
Profesor lector

Ing. Gabriela Ortiz León
Profesor lector

Ing. Eduardo Interiano Salguero
Profesor asesor

Los miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica

Cartago, Octubre del 2010.

Declaro que el presente Proyecto de Graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado bibliografía, he procedido a indicar las fuentes mediante las respectivas citas bibliográficas.

En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

Cartago, Noviembre 2010

Manuel Leiva Fernández.
Céd.: 303950944

Resumen

Actualmente empresas desarrolladoras de dispositivos electrónicos crean una variedad de *system on chip,* (SoC) procurando ofrecer una solución a cada una de las aplicaciones que se encuentran en el mercado y satisfacer la demanda generada. Estos sistemas empotrados ofrecen diferentes tipos de periféricos, procesador de propósito específico para el procesamiento digital de señales, para uso en aplicaciones de audio, video o uso general en la industrial.

Para ofrecer soporte a estos dispositivos, comunidades en el mundo (empresas y personas) se han unido para crear soluciones de *software* abiertas (*open software*) como programas libres, herramientas de desarrollo para sistemas empotrados o bibliotecas de *software*, las cuales pueden ser utilizadas para crear aplicaciones específicas a bajo costo y con un gran apoyo de la comunidad de código abierto.

Este trabajo muestra el desarrollo de una biblioteca para uso en aplicaciones de control automático en la tarjeta *open hardware* Hawkboard con capacidad para el uso del procesador digital de señales. El código de la biblioteca es abierto para su optimización y desarrollo. Se utilizan solo herramientas de *software* libre para la creación de la biblioteca.

Palabras Clave

Control Automático, Biblioteca de *software*, DSP, Hawkboard, Linux, OMAP-L138

Abstract

Nowadays companies producing electronic devices provide a variety of systems on chip (SoC) trying to offer a solution for each of the applications on the market and so to meet the demands generated. These embedded systems offer a wide choice of peripherals, specific purpose processor for digital signal processing for use in audio and video applications or general industrial purposes.

To provide support for these devices, communities around the world (companies and people) have teamed up to create open software solutions to the world, as free programs, development tools for embedded systems or software libraries, which can be used to create specific applications at low cost and with great support from the open source community.

This work shows the development of a library to be used in automatic control applications on the open hardware board Hawkboard, with capacity to use the digital signal processor. The library code is open for optimization and development. Only free software tools were used in the creation of the library.

Keywords

Automatic control, DSP, Hawkboard, Linux, OMAP-L138, software library

Dedicatoria

Dedico este trabajo a mi familia por el apoyo constante y su amor incondicional, en especial a mis padres Norma Fernández Ramírez y Gerardo Leiva Granados por su confianza, comprensión, los consejos brindados y por ser la más grande guía durante todas las etapas de mi vida. Finalmente agradezco a mis hermanos María y José por su apoyo y acompañarme en todo momento.

Gracias.

Manuel Leiva Fernández

Agradecimiento

Primero deseo dar gracias a Dios por permitirme vivir esta etapa y darme fortaleza durante todos mis años de carrera universitaria.

Agradezco al Profesor Ing. Eduardo Interiano S. por su confianza en mí para realizar el proyecto. Al profesor Dr. Pablo Alvarado M. por todos los conocimientos enseñados, su dedicación y motivación durante el desarrollo del trabajo.

Gracias a Gerardo Leiva y a Lohana Meneses por su actitud crítica y por todo el tiempo dedicado a la lectura de este documento.

ÍNDICE GENERAL

ÍNDICE FIGURAS

ÍNDICE TABLAS

Capitulo 1. Introducción.

En la carrera de Ingeniería Electrónica, se encuentra el curso: Control Automático, en el cual el estudiante se forma con el conocimiento necesario para poder implementar reguladores y compensadores para sistemas de control.

Este curso teórico se complementa con el curso: Laboratorio de Control Automático, en el cual un grupo de estudiantes tiene que desarrollar un proyecto único. Éste consiste en el desarrollo completo de un proyecto de control que abarca, desde la definición del problema a resolver a la obtención del modelo de la planta, para luego realizar el diseño de la solución, la síntesis e implementación del regulador.

El Laboratorio de Control Automático cuenta con varios sistemas o plantas que han sido desarrolladas a través de los años con materiales suministrados por los mismos estudiantes que en su momento utilizaron el sistema y por la Escuela de Electrónica, la cual destina aproximadamente $500 dólares por semestre al laboratorio para la compra de componentes nuevos y remplazo de dispositivos dañados.

Usualmente cada planta debe ser acondicionada por el grupo de estudiantes debido a que el estudiante actúa como diseñador y goza de libertad a la hora de diseñar la solución; pero, cumpliendo con las especificaciones dadas. Así el estudiante llega a conocer con profundidad el modelo de su planta, logrando optimizar la solución. Un punto negativo de esta metodología es que los conocimientos obtenidos por el estudiante, son con base a la utilización de una sola planta, por lo que no puede conocer las particularidades de otros sistemas.

Una alternativa al problema anterior es la elaboración de proyectos cortos donde el estudiante puede realizar un regulador adecuado para diferentes plantas, pero esto conlleva realizar la interfaz necesaria para la toma de datos, acople entre el

controlador a implementar y la planta. Además se debe de contar con una estructura de control específica para cada planta, lo cual hace que sea difícil la implementación de múltiples sistemas dentro de un periodo de 16 semanas.

Hoy en día la tecnología ha permitido el uso de nuevas herramientas académicas. Esto permite al profesor poder poner a disposición del estudiante herramientas de análisis teórico-práctico, las cuales servirán de apoyo y complemento a la clase normalmente desarrollada.

El proyecto TeleLab desarrollado por el Profesor Eduardo Interiano, tiene el objetivo de crear un laboratorio en línea, el cual dotará a los estudiantes de una plataforma de desarrollo donde ellos puedan acceder a diferentes experimentos, proporcionando opciones como: tomar mediciones, crear su propio controlador y este ser aplicado a la planta utilizada para el respectivo experimento, permitiendo al estudiante poder desarrollar los experimentos remotamente desde una computadora a través de la red de internet [2]. Esto permite que el estudiante dedique su trabajo a realizar el modelo de la planta y el regulador necesario para su control debido a que no tiene que destinar tiempo en la elaboración e implementación de la estructura de *hardware* necesaria para el controlador del sistema, tales como puertos de control, sensores, filtros y unidad de control programable permitiendo enfocarse en el cumplimiento y desarrollo de los objetivos específicos que conforman el Laboratorio de Control Automático.

La importancia de este sistema no solo reside en su uso práctico dentro del laboratorio, sino que uno de los objetivos del proyecto es generar conjuntamente conocimiento a partir de la investigación hecha para continuar con nuevos desarrollos tales como: *Hardware in the loop* (el cual consiste en un sistema electrónico que simula el comportamiento de una planta ante las señales de control que se le son aplicadas), desarrollo de *hardware* para implementación en aplicaciones de control automático, código abierto que permita ser utilizado,

desarrollado y optimizado por los mismos estudiantes permitiendo la transmisión de conocimientos.

Este proyecto pretende crear una solución de *software* abierto para aplicaciones de control automático en un sistema empotrado, utilizando un sistema operativo basado en un kernel GNU/Linux.

Se realiza la creación de una biblioteca de *software*, la cual contiene estructuras y algoritmos para aplicaciones en control automático de procesos, para uso en sistemas empotrados, utilizando para la implementación del proyecto la plataforma Hawkboard.

Capitulo 2. Meta y Objetivos.

2.1 Meta.

Emplear el proyecto con el fin de convertirlo en una herramienta escalable, flexible y abierta de *software* y *hardware* empotrado el cual será utilizado como parte del Laboratorio de Control Automático para el uso por parte de los estudiantes.

2.2 Objetivo General.

Desarrollar una biblioteca de *software* empotrado para la utilización de algoritmos aplicados a sistemas de control automático y procesamiento general de señales.

2.3 Objetivos Específicos.

1. Desarrollar una biblioteca que permita el uso de módulos de control automático para su implementación en el procesador de propósito general (ARM9) de la plataforma Hawkboard.

2. Implementar una estructura de programación que permita ejecutar módulos de control automático en el DSP C674. (*Floating Point DSP*)

3. Crear una aplicación para demostrar las funciones y capacidades de la biblioteca.

Capitulo 3. Marco Teórico

3.1 Antecedentes

3.1.1 Unidad controladora de procesos

La Unidad Controladora de Procesos para la implementación de sistemas de control automático a través de redes TCP/IP es un proyecto creado como parte del proyecto TeleLab. Este sistema permite la ejecución de sistemas de control, remotamente a través de la red Ethernet permitiendo el diseño, análisis, simulación de los sistemas y su ejecución utilizando plantas reales.

El sistema utiliza procesador microchip dsPIC33FJ256MC710 para procesamiento de los datos y un dispositivo de acceso remoto Lantronix, que permite acceso a una computadora y control del dispositivo remotamente [2].

3.1.2 Entorno de desarrollo integrado

Este es un proyecto que da continuación al proyecto de Unidad Controladora de Procesos. Este proyecto desarrolló un sistema de interacción con el usuario denominado Entorno de Desarrollo Integrado para una Unidad Controladora de Procesos, el cual consiste en el diseño de una interfaz gráfica, un traductor de lenguaje gráfico, el cual permite comunicar el entorno gráfico con la unidad y la implementación de la herramienta de simulación. Toda la interfaz se realiza por medio de una aplicación (*applet*) cargada en una página web la cual permite la creación de sistemas de control gráficamente con opción de ejecutarlo remotamente en la unidad o simularlo según lo requiera el usuario [1].

3.2 Hawkboard

3.2.1 Descripción de la tarjeta Hawkboard

Hawkboard es una tarjeta *Open Community*, diseñada para proporcionar a la comunidad una solución rica en características y económica, basada en el procesador de *Texas Instrument* (TI) OMAP-L138. Esta tarjeta contiene una amplia gama de periféricos.

El OMAP-L138 ofrece un ARM9 integrado y un DSP TI. Su bajo nivel de consumo de energía de este sistema lo abre a múltiples aplicaciones integradas e industriales.

El controlador de memoria interna ofrece soporte para memorias como DDR2/MDDR/SDRAM/NOR y memoria flash NAND.

Un controlador SATA disponible para el soporte de las interfaces SATAI SATA II lo cual permite ampliar la capacidad de almacenamiento de esta plataforma.

Un controlador MMC/SD incorporado para proporcionar almacenamiento para guardar los archivos personales.

Un puerto UPP (*Universal Parallel Port*) Es una interfaz paralela de alta velocidad para conexión a FPGA y convertidores de datos.

Dos puertos USB proporcionan gran variedad de conectividad de periféricos. El puerto USB OTG también proporciona una opción para alimentar la tarjeta cuando se conecta a una PC o portátil [7].

La tarjeta Hawkboard (Figura 3.1) provee al usuario los periféricos mostrados en la Tabla 3.1.

Tabla 3.1 Características y periféricos de la tarjeta Hawkboard.

Núcleo
C674x, ARM9
Frecuencia: 375,456 MHz
Memoria
128 MByte DDR2 SDRAM de 150MHz.
128 MByte NAND FLASH
1 slot SD/MMC
Interfaces
Un puerto serie RS232
Un puerto Fast Ethernet (10/100 Mbps)
Un puerto USB *Host* (USB 1.1)
Un puerto USB OTG (USB 2.0)
Un puerto SATA (3Gbps)
Un puerto VGA (15 pin D-SUB)
Dos Puertos de AUDIO (1 LINE IN & 1 LINE OUT)
Un puerto de video compuesto IN (RCA Jack)
Interfaz de expansión
VPIF, UPP, PRU, LCDC, UART(x2) SPI(x2), I2C (x1), eCAP, eHRPWM, GPIO

Figura 3.1 Hawkboard [7].

3.2.2 OMAP L-138

3.2.2.1 Descripción

Este dispositivo es un procesador para aplicaciones de baja potencia conformado por un procesador ARM926EJ-S y un DSP C674x. El cual posee un rendimiento en potencia mucho mayor que otros miembros de la plataforma DSP TMS320C6000.

El dispositivo permite a los OEM (*Original Equipment Manufacturer*) y ODM (*Original Design Manufacturer*) llevar rápidamente dispositivos al mercado caracterizados con un fuerte soporte en sistemas operativos, variedad en interfaces de usuario y larga vida en rendimiento de procesamiento a través de la máxima

flexibilidad de una solución completamente integrada de procesador y DSP en conjunto [22].

3.2.2.2 El ARM926EJ-S

El procesador ARM926EJ-S es un miembro de la familia de los procesadores ARM9 de propósito general. Este procesador está dirigido a aplicaciones multitarea en la gestión de memoria completa, de alto rendimiento.

El procesador ARM926EJ-S tiene una arquitectura Harvard (arquitectura con caminos físicos separados para instrucciones y datos) y proporciona un subsistema completo de alto rendimiento. El ARM926EJ-S es un procesador RISC de 32 bits que realiza instrucciones de 32 bits o 16 bits y procesamiento de datos en 32 bits, 16 bits y 8 bits. El núcleo utiliza segmentación (*pipeline*) lo cual permite que todas las partes del procesador y memoria del sistema puedan funcionar de forma continua [22].

3.2.2.3 C674x DSP

La Unidad Central de Procesamiento C674x (CPU) consta de ocho unidades funcionales, dos rutas de datos y dos registros de propósito general. Ambos registros (*register file* A y *register file* B), contienen cada uno 32 registros de 32 bits para un total de 64 registros. Los registros de propósito general se pueden utilizar para los datos o pueden ser punteros a datos. Los tipos de datos soportados incluyen paquetes de datos de 8, 16, 32, 40 y 64 bits [22].

Las 8 unidades funcionales (.M1, .L1, .D1, .S1, .M2, .L2, .D2, and .S2) tienen capacidad cada una para ejecutar una instrucción cada ciclo de reloj.

La unidad .M realiza operaciones de multiplicación. Estas operaciones pueden aplicarse de forma múltiple según la longitud del dato, por ejemplo la unidad .M puede realizar una de las siguientes operaciones por ciclo de reloj: una multiplicación de 32 x 32 bits, cuatro multiplicaciones de 16 x 16 bits con capacidad de suma y

resta o una multiplicación compleja de 16 x 16 bits produciendo una salida de 32 bits para la parte real y 32 bits para la parte imaginaria.

La unidad .L incorpora habilidad para hacer suma y resta en paralelo.

La unidad .D primeramente carga datos de la memoria hacia los registros y el resultado lo almacena desde los registros hacia la memoria.

El DSP C674x combina el rendimiento del núcleo C64x+ con las capacidades de punto flotante del nucleoC67x+.

3.3 Linux

3.3.1 Linux Kernel

El kernel es el componente central de la mayoría de sistemas operativos, es un puente entre las aplicaciones y el procesamiento de datos a nivel de *hardware*. Es el encargado de que el *software* y el *hardware* puedan trabajar juntos en el sistema [12].

Las funciones más importantes del mismo son:

- Administración de la memoria para todos los programas y procesos en ejecución.
- Administración del tiempo de procesador que los programas y procesos en ejecución utilizan.
- Es el encargado de acceder a los periféricos y elementos del ordenador para dar soporte a las aplicaciones.

3.3.2 *Universal bootloader*

Universal bootloader (Uboot) es un programa diseñado exclusivamente para preparar todo lo que necesita el sistema operativo para funcionar. Este programa al ejecutarse carga el sistema operativo desde una unidad de almacenamiento hacia la memoria RAM para que pueda ser ejecutado por el procesador [4].

Uboot ofrece comunicación con el usuario a través de consola por medio de instrucciones y el establecimiento de variables de ambiente. Esta comunicación puede automatizarse y ser realizada a través de un equipo de desarrollo de *software* (Ver sección 3.4).

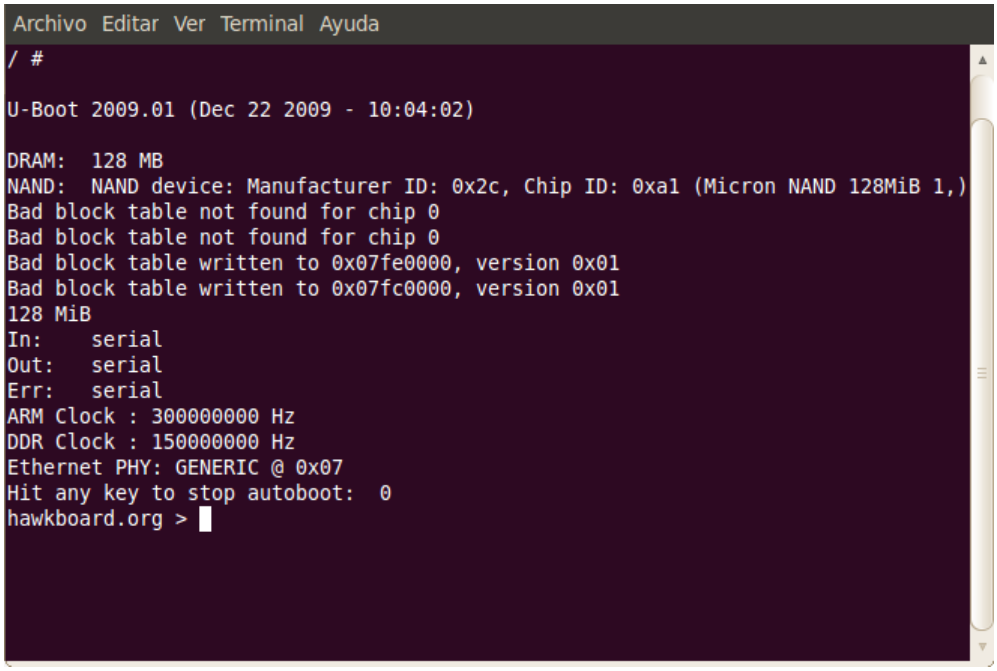

Figura 3.2 Carga del Uboot en la plataforma Hawkboard.

3.4 Kit de desarrollo de *software* para Hawkboard

3.4.1 SDK

Un equipo de desarrollo de *software* o SDK (*Software Development Kit*) es generalmente un conjunto de herramientas de desarrollo que le permite a un programador crear aplicaciones para un sistema concreto.

3.4.2 RidgeRun SDK

RidgeRun SDK es un kit de desarrollo, creado por RidgeRun *Embedded Solutions* para propósitos de evaluación. Una licencia libre está disponible para uso académico o personal [14].

RidgeRun SDK ofrece entre sus características más relevantes:

- Integración completa en un menú principal de toda la configuración del SDK. Esto permite el acceso a todos los menús de configuración para la cadena de herramientas, *bootloader*, sistema de archivos y otros desde un punto centralizado.
- Manejo transparente de las dependencias de la arquitectura. Con el fin de que cambios realizados en *hardware*, *toolchain* o *bootloader*, es simplemente seleccionar la configuración correcta en el menú del SDK principal y todas las dependencias necesarias serán manejadas por el SDK.
- Fácil integración de aplicaciones de usuario nuevas.

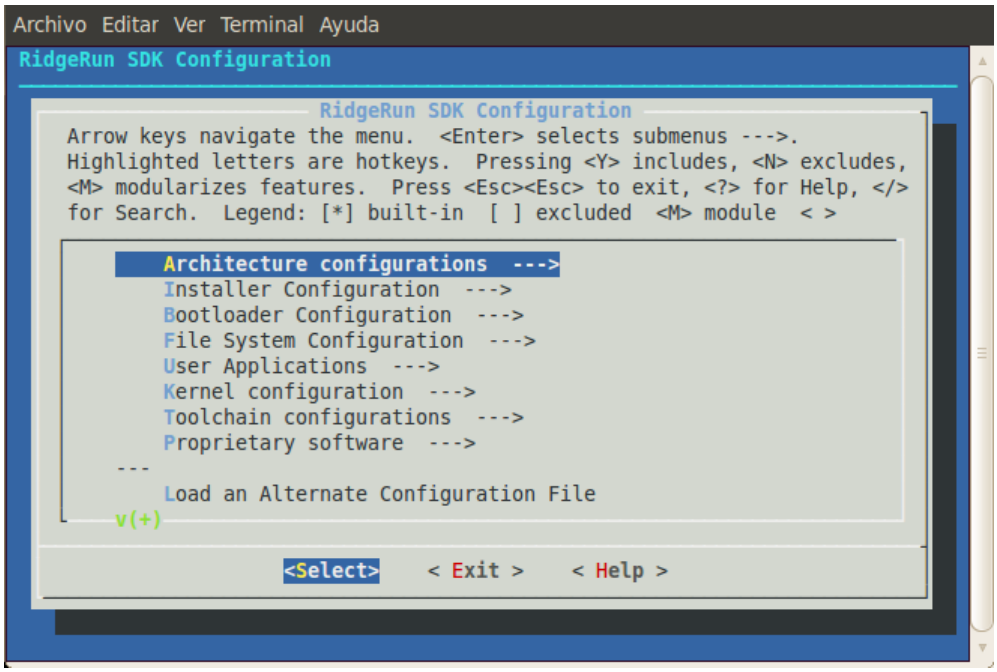

Figura 3.3 Menú de configuración del SDK de RidgeRun.

El SDK incluye los siguientes paquetes:

- Bootloader: u-boot-2009.01.
- Kernel: Linux-2.6.33-
- Toolchain: Arm-linux-gnueabi 2008.
- *Digital Video Software Develoment Kits.* (DVSDK.2.00.00) Incluye herramientas como *Codec Engine*, xDAIS, DSPLink.

3.5 Herramientas para el desarrollo de *software*

3.5.1 *Toolchain*

Es un conjunto de herramientas y utilidades que se usan para crear aplicaciones para una determinada plataforma, permitiendo una compilación cruzada. GNU *Toolchain* para procesadores ARM contiene herramientas de línea de comando para la creación y compilación cruzada de una aplicación, entre las herramientas

más importantes incluye, compilador GNU C y C++, GNU *assembler*, *linker* y *archiver* C y C++ *runtime libraries* y GNU *debugger* [3].

3.5.2 Aplicación *Make*

La aplicación *make* es una herramienta de generación o automatización de código, determina automáticamente qué piezas de un programa se tienen que recompilar, y las sentencias necesarias para realizar la recompilación.

El programa *make* utiliza un archivo de instrucciones (*script*) escrito por el usuario llamado usualmente *Makefile* el cual tiene una sintaxis propia de la aplicación *make*.

Un esquema general de la sintaxis del archivo se observa en la Figura 3.4.

```
target ... : prerequisites ...
  recipe
  ...
  ...
```

Figura 3.4 Sintaxis general del archivo Makefile.

target: suele ser el nombre de un archivo que se genera por un programa, ejemplos de objetivos son los archivos ejecutables o un objeto.

prerequisites: es un archivo que se utiliza como entrada para crear el blanco. Un objetivo a menudo depende de varios archivos.

recipe: es la acción que se lleva a cabo, puede tener más de una sentencia, ya sea en la misma línea o cada una en su propia línea.

3.5.3 Doxygen

Doxygen es una herramienta para la generación automática de documentación de código. Doxygen permite generar documentación como [5]:

a. Puede generar documentación on-line (en HTML) y o un manual de referencia *off-line* (en LaTeX) de un conjunto de archivos de código fuente documentados. También hay soporte para generar salida en RTF (MS-Word), *PostScript*, PDF con hipervínculos, HTML comprimido, y las páginas man de Unix.
b. La documentación se extrae directamente de las fuentes, lo cual hace mucho más fácil mantener la documentación de conformidad con el código fuente.
c. Doxygen se puede configurar para extraer la estructura del código de los archivos fuente no documentados. Esto es muy útil para encontrar rápidamente caminos en distribuciones de código muy grandes. También puede visualizar las relaciones entre los diversos elementos incluyendo gráficos de dependencia, diagramas de herencia y diagramas de colaboración, los cuales son generados de forma automática.

3.5.4 Repositorio

Repositorio es un lugar local o remoto que contiene toda la información que permite obtener las versiones anteriores de los archivos de código fuente en cualquier momento. Existen diferentes herramientas para el uso automático del repositorio tales como *Subversion* y GIT.

Subversion es un sistema de control de versiones, lo que permite mantener las versiones anteriores de los archivos y directorios de código fuente, mantener un registro de quién, cuándo y qué cambios se produjeron. *Subversion* mantiene una sola copia del código fuente en el repositorio. El usuario puede editar el código en un lugar llamado copia de trabajo y luego exportar los cambios hacia el repositorio [15].

El uso de *Subversion* requiere de los siguientes pasos básicos.

1. Creación del repositorio. (svn mkdir)
2. Importar archivos en el repositorio.(svn import)

3. Crear una copia de trabajo.(svn checkout)
4. Actualizar la copia de trabajo.(svn update)
5. Exportar los cambios realizados a la copia de trabajo.(svn commit)

Una vez creado el repositorio solo son necesarios los pasos 4 y 5 para trabajar en la copia de trabajo.

3.6 Biblioteca estática

Es una colección de rutinas o subrutinas usadas para desarrollar *software*. Es una colección de archivos de objetos, convencionalmente el final del nombre de la biblioteca termina con el sufijo “.a”. Esta colección se crea con la herramienta “ar” (archivador) del programa.

Bibliotecas estáticas permiten a los usuarios conectarse a los programas sin tener que recompilar su código, ahorrando tiempo recompilación. En teoría, el código de bibliotecas ELF estáticas que están integradas en un archivo ejecutable debe correr un poco más rápido (entre uno 1 y 5%) que en una biblioteca compartida o una biblioteca de carga dinámica. Estas bibliotecas son en especial útiles en sistemas empotrados, cuando la aplicación quiere ser ejecutada en varios sistemas, debido a que una vez compilada la aplicación es completamente autónoma y no requiere que la biblioteca se encuentre en cada uno de los sistemas para ejecutarse [13].

3.7 Archivos ejecutables

3.7.1 Executable and Linkable Format File

*Executable and Linkable Format File (*ELF) es un formato de archivo utilizado en sistemas GNU Linux, este formato es utilizado en ejecutables, código objeto y bibliotecas.

Los archivos ELF contienen una cabecera común para las diferentes estructuras. Una sección especial llamada ELF *Header* contiene entre su información el tipo de archivo y la arquitectura. Esto permite que el archivo ELF pueda ser identificado por el sistema operativo indiferentemente a la arquitectura para la que se creó.

Por ejemplo la Figura 3.5 muestra el valor de las variables *Type* y *Machine* de la sección *Header* de dos archivos llamados “pcu” compilados para las arquitecturas Intel y ARM (readelf es un programa en Linux que permite ver información de las diferentes secciones de un archivo ELF).

```
Readelf bin/pcu –h
ELF Header:
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
readelf bin/pcu -h
ELF Header:
  Type:                              EXEC (Executable file)
  Machine:                           ARM
```

Figura 3.5 Variables Type y Machine de la sección header del ELF para la arquitectura ARM e INTEL.

3.7.2 *Application binary interface*

También llamado ABI describe la interfaz a bajo nivel entre la aplicación y el sistema operativo. ABI también define el formato binario de archivos objeto y bibliotecas. El estándar para sistemas empotrados es llamado, *Embedded-Application Binary Interface* (EABI).

EABI permite que un compilador que soporte este estándar, cree código objeto compatible con otros compiladores; así permite que los desarrolladores utilicen bibliotecas generadas con un compilador diferente al utilizado por el desarrollador de la biblioteca [24].

3.8 Herramientas de desarrollo para el uso del DSP

3.8.1 *Codec Engine*

Codec Engine (CE) es un entorno (*framework)* que permite a las aplicaciones instanciar *codecs* y algoritmos xDAIS (eXpressDSP *Algorithm Standard*) usando un API común y posibilita al usuario ejecutar código en el DSP y llamarlo desde el GPP ARM [17].

El API permite:

- Que el algoritmo se pueda ejecutar de forma local (en el GPP) o remoto. (en el DSP)
- El sistema puede ser un GPP + DSP, el DSP solamente, sólo GPP.

3.8.2 C6Run

3.8.2.1 Descripción

C6Run es una herramienta gratuita de desarrollo *Open Source* de *Texas Instruments* que permite al usuario utilizar sin problemas el DSP en dispositivos heterogéneos ARM + DSP. Este proyecto está destinado a ser utilizado en un sistema que corra en ARM Linux y que contengan un núcleo DSP C6000.

El objetivo de este proyecto es facilitar el desarrollo inicial y la carga de código en el DSP para los desarrolladores ARM que están familiarizados con la creación de aplicaciones para el sistema operativo Linux con un ARM GCC.

C6Run es un conjunto de herramientas que generan ejecutables o bibliotecas en la arquitectura ARM a partir de los archivos C y permiten potenciar el DSP para ejecutar el código C.

Hay dos usos del proyecto C6Run, expuestos a través de dos diferentes *front-end scripts* llamados C6RunLib y C6RunApp [19].

El sistema de construcción tiene varias dependencias. Cuando C6Run se encuentra dentro de un producto oficial de TI SDK, estos componentes son siempre provistos. Cuando se utiliza como un producto independiente, el archivo makefile de nivel superior en el paquete se puede utilizar para recuperar automáticamente todos los componentes necesarios, con excepción del compilador de DSP y ARM

C6Run utiliza las siguientes dependencias: DSP/BIOS, DSPLink, XDCtools Linux Utils, Local Power Manager (LPM), TI C6000 Codegen Tools, ARM Cross Compiler tools.

C6Run brinda soporte a las siguientes plataformas:

- Hawkboard
- BeagleBoard
- BeagleBoard-xM
- OMAP 3530
- OMAP L137
- OMAP L138

3.8.2.2 C6RunLib

C6RunLib es utilizado para construir una biblioteca estática que pueda ser vinculada con una aplicación ARM y facilitar el acceso al DSP cuando las funciones de la biblioteca son llamadas. Esto permite al usuario mantener partes de la aplicación en el ARM mover otras partes de la DSP.

C6RunLib se compone de dos secuencias de instrucciones, c6runlib-cc y c6runlib-ar. La herramienta c6runlib-cc se utiliza para compilar el código C para crear archivos objeto C6000 DSP. La herramienta c6runlib-ar es un archivador especializado que

toma los archivos objeto DSP producido por la herramienta de c6runlib-cc y genera una biblioteca. Esta biblioteca de salida puede estar vinculada a la aplicación de ARM para proporcionar acceso transparente a todo el DSP a través de las interfaces de la biblioteca. La aplicación se ejecuta desde el ARM como una aplicación, sin embargo debajo de la aplicación, el DSP se inicia, se carga con el código del programa (que incluye las funciones de la biblioteca), y espera las llamadas a funciones desde el ARM para ejecutar como lo muestra la Figura 3.6. C6RunLib extrae las funciones pertinentes para ejecutar en el DSP [20].

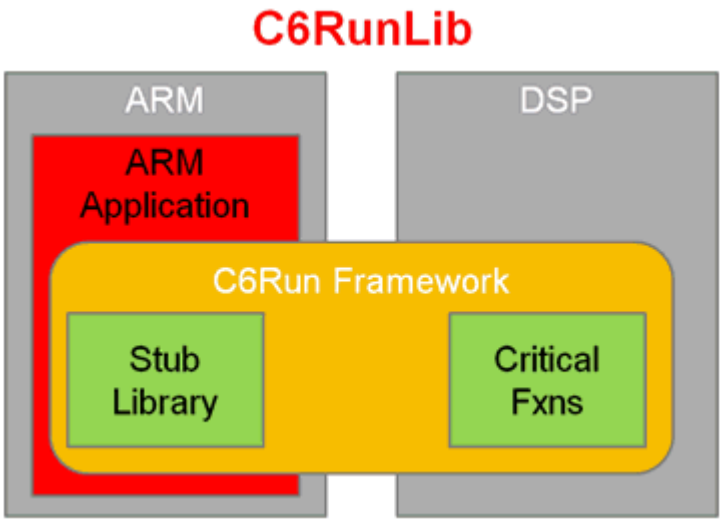

Figura 3.6 C6RunLib Framework

3.8.2.3 C6RunApp

Esta herramienta funciona como un compilador cruzado para el DSP. Permite que aplicaciones en C sean reconstruidas para el núcleo C6000 DSP de *Texas Instruments*. El frente C6RunApp se compone de un comando, llamado c6runapp-cc. El código C se compila para generar archivos objeto C6000 y los enlaza en una aplicación. Al realizar las operaciones de unión, la herramienta hace uso de una serie de pasos (incluidos los enlaces con las herramientas de generación de código C6000) para crear un ejecutable en ARM a partir de los archivos objeto del DSP.

Una vez construida, la aplicación se puede ejecutar desde el ARM, como si fuera una aplicación nativa del ARM, sin embargo debajo de la aplicación, el DSP se inicia, carga el código del programa, y ejecuta la aplicación hasta su finalización. Esto hace parecer a la aplicación como otro proceso del ARM, pero todo el código C compilado utilizando esta herramienta se ejecuta en el núcleo DSP como lo observa en la Figura 3.7 [19].

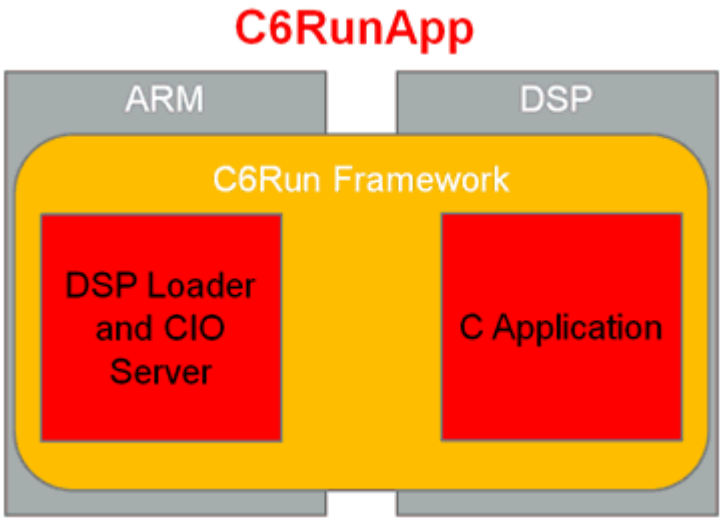

Figura 3.7 C6RunApp Framework.

3.8.3 Dsp/Bios Link

Es un *software* para la comunicación inter procesador (GPP-DSP). Proporciona un API genérico que resume las características del enlace físico que conecta el GPP y el DSP [21].

Dependiendo del soporte para la plataforma y el sistema operativo, DSP/BIOS LINK proporciona un subconjunto de los siguientes servicios a sus clientes:

- Control básico del procesador.
- Memoria compartida sincronizada a través de múltiples procesadores.
- Notificación de eventos de usuario.
- Se excluye mutuamente el acceso a la estructura de datos compartidos.

- Lista de enlace basada en el flujo de datos.
- Transferencia de datos a través de canales lógicos.
- Mensajería (basado en el módulo MSGQ de DSP/BIOS).
- Buffer de anillo basado en el flujo de datos.

Una aplicación típica puede no requerir todos los servicios prestados por DSP/BIOS LINK, un uso típico puede ser usar sólo como mecanismo para transferir mensajes entre GPP y DSP.

3.8.4 CMEM

CMEM es un API para el manejo de uno o más bloques de memoria contigua físicamente. También proporciona servicios de traducción de direcciones (por ejemplo de virtuales a la traducción física) y la gestión de memoria caché. Esta memoria físicamente contigua es útil como búfer de datos que será compartido con otro procesador (por ejemplo, para el DSP y el ARM) o un acelerador de *hardware* DMA. A través de su configuración, CMEM permite a los usuarios evitar la fragmentación de memoria, y se asegura que grandes bloques de memoria físicamente contigua estén disponibles permanentemente mientras el sistema este en funcionamiento [16].

3.9 Controladores y filtros digitales

3.9.1 Controlador digital

Es un mecanismo que se utiliza en sistemas de control industriales. La Figura 3.8 muestra un controlador PID utilizado aplicaciones industriales, el cual actúa sobre la variable a ser manipulada mediante la combinación de tres acciones de control: proporcional, integral y derivativa [9].

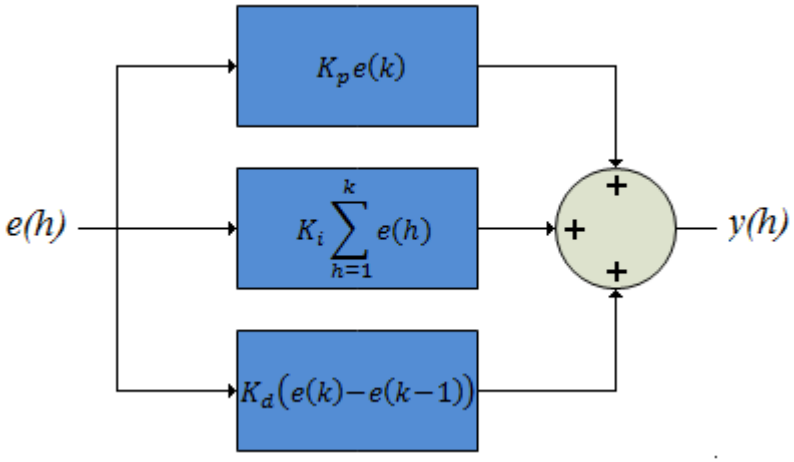

Figura 3.8. Controlador PID.

3.9.2 Filtro digital

Es un algoritmo de cálculo que procesa una señal digital permitiendo el paso de algunas componentes de frecuencia deseadas.

El filtro FIR (*Finite Impulse Response*) es un tipo de filtro que ante una entrada de una función impulso la salida tendrá un número finito de números no nulos. Este filtro es también llamado filtro no recursivo debido a que el valor de la salida solo depende de los valores pasados y presentes de la entrada [9].

La función de transferencia se expresa como:

$$\frac{Y(z)}{X(z)} = b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_m z^{-m} \qquad (3.1)$$

La Figura 3.9 representa un filtro FIR en forma de diagrama de bloque:

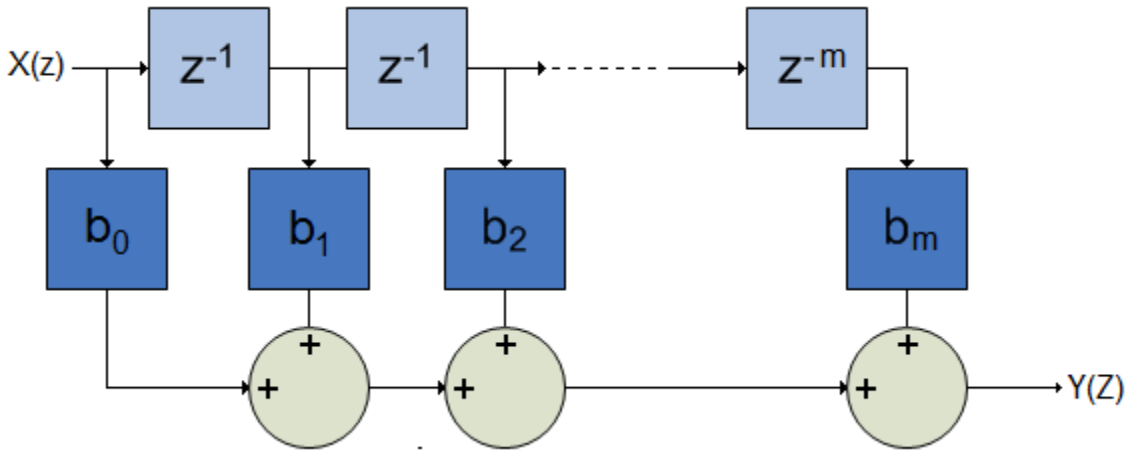

Figura 3.9 Diagrama de bloques del filtro FIR.

Capitulo 4. *Process Controller Unit* (PCU)

4.1 Descripción

Esta biblioteca se realizó con el fin de ofrecer una plataforma de *software* mínima para el uso en sistemas empotrados, utilizando la tarjeta Hawkboard, como plataforma para la implementación de la biblioteca.

La biblioteca cuenta con diferentes niveles de abstracción para el uso de los módulos, para simplificar la interacción del usuario con el sistema. Lo anterior permite que un su nivel más bajo de abstracción el código fuente pueda ser utilizado en otras plataformas con poca capacidad para la ejecución de *software* y en el nivel superior de abstracción proporciona al usuario funciones para el manejo general de las estructuras sin importar el tipo (todos los bloques de control son vistos como un mismo tipo de datos). La biblioteca esconde los detalles específicos de cada módulo de cómo se almacenan y mantienen los datos y ofrece funciones para que el sistema sea manejable.

La biblioteca se encuentra en un repositorio y debe ser descargada (para la descarga se utiliza la herramienta *Subversion*) y se ubicará en un directorio elegido por el usuario el cual se le llamará de forma estándar PCUDIR.

La biblioteca establece una serie de variables de ambiente dentro del *bash* (intérprete de instrucciones de Linux) entre las cuales se encuentra PCUDIR, esta variable contienen la ruta hacia el directorio de la biblioteca, lo cual permite al usuario conocer la ruta de acceso a la biblioteca, permitiendo obtener información de la biblioteca desde cualquier punto del sistema de archivos. Por ejemplo para observar los archivos dentro del directorio de la biblioteca desde cualquier directorio escriba [10]:

```
ls $PCUDIR
bin  doc  include  lib  Makefile  Rules.make  src
```

Este documento utilizará PCIDIR para referirse a la ruta del directorio de la biblioteca PCU.

4.2 Estructura general de la biblioteca

La estructura general está conformada por cinco directorios dentro del directorio general de la biblioteca. La estructura en general hace una división entre los archivos de la biblioteca según el tipo y función que realiza.

La Figura 4.1 muestra todos los directorios que conforman la biblioteca.

```
tree $PCUDIR -d
|-- bin
|-- doc
|   |-- documentation
|   `-- images
|-- include
|-- lib
|   `-- dsplib
`-- src

8 directories
```

Figura 4.1 Árbol de directorios que conforman la biblioteca.

La descripción de cada directorio se realiza a continuación.

bin: (*binary*) contiene el archivo ejecutable de la aplicación, este directorio también puede contener algún otro archivo que utilice la aplicación tal como archivos de configuración.

src: (*source*) este directorio es para uso exclusivo del usuario, contiene los archivos fuente de la aplicación creada por el usuario, cuando se compila los archivos de código fuente y se genera la aplicación, ésta se crea en el directorio bin/.

lib: (*library*) contiene todos los archivos de código fuente (*.c) para crear la biblioteca. Las bibliotecas generadas también son guardadas en este directorio. Este directorio posee un subdirectorio llamado dsplib/, el cual contiene archivos de código fuente para la generación de bibliotecas para ejecutar en el DSP.

include: contiene todos los archivos cabecera (*.h) con los prototipos de las funciones de la biblioteca.

doc: (*documentation*) contiene toda la documentación de la biblioteca. Incluye la documentación del código fuente creada por la aplicación Doxygen. Este directorio contiene varios subdirectorios, descritos a continuación:

documentation: contiene guías de usuario generadas por el desarrollador de la biblioteca para su utilización.

images: este directorio contiene las imágenes utilizadas para la documentación del código de la biblioteca.

4.3 Construcción de la biblioteca

La construcción se realiza por medio de la aplicación *Make*, esta herramienta utiliza un archivo llamado Makefile el cual determina los archivos fuente necesarios para la generación de la biblioteca y aplica las instrucciones necesarias para realizar cada acción.

Un Makefile es capaz de ejecutar múltiples acciones, éstas son llamadas objetivos (*target*) dentro del estándar del Makefile.

La biblioteca contiene múltiples archivos Makefile, los cuales tienen un nivel de jerarquía según su ubicación y propósito como lo muestra la Figura 4.2.

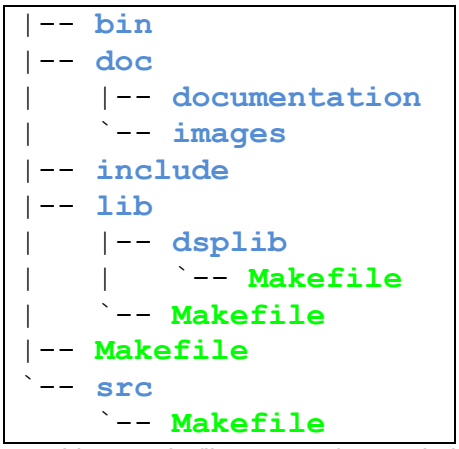

Figura 4.2 Archivos Makefile que conforman la biblioteca.

Los archivos Makefile son descritos a continuación según su ubicación.

$PCUDIR/Makefile: Este es el archivo Makefile principal, se encuentra en el nivel superior del proyecto, a través de éste, es posible realizar la construcción general de la biblioteca. Las principales funciones que realiza son:

a. Establece las variables de ambiente necesarias por la biblioteca para la compilación.
b. Permite crear la biblioteca o una aplicación creada por el usuario.
c. Crea la documentación del código.

$PCUDIR/lib/Makefile: Realiza la creación de todas las bibliotecas para el GPP.

$PCUDIR/lib/dsplib/Makefile: Realiza la creación de las bibliotecas de propósito específico en el DSP.

$PCUDIR/src/Makefile: Compila y crea el archivo ejecutable de la aplicación. Este Makefile determina el nombre del archivo con código fuente en C y lo compila enlazando todas las bibliotecas de forma automática lo cual permite al usuario, que solo deba incluir el archivo cabecera de la biblioteca a utilizar.

Una vez establecidas las variables de ambiente por el Makefile principal, cada uno de los Makefile pueden ser ejecutados de forma independiente en el directorio en el que se encuentran, por ejemplo para realizar la construcción de la biblioteca, se puede cambiar al directorio $PCUDIR/lib/ y ejecutar el Makefile en lugar de hacerlo a través del Makefile principal.

Todos los Makefile comparten un conjunto de objetivos generales mostrados en la Tabla 4.1.

Tabla 4.1 Objetivos generales en cada Makefile.

Objetivo	Descripción
all	Realiza la compilación completa de los archivos en el directorio.
clean	Elimina todos los archivos que son generados al compilar el directorio
clean-object	Elimina los archivos objeto generados al compilar una biblioteca.
help	Muestra una ayuda de los objetivos disponibles en el Makefile.

Todos los *Makefile* permiten la aplicación una serie de parámetros en el momento de realizar la compilación, los parámetros son mostrados en la Tabla 4.2.

Tabla 4.2 Parámetros de cada Makefile.

Parámetro	Descripción
DEBUG=1	Produce información de depuración.
TARGET=ARM	Realiza la compilación para el procesador ARM.
VERBOSE=1	Muestra las instrucciones ejecutadas para cada objetivo del Makefile.

Todos los parámetros pueden ser aplicados de forma conjunta.

4.4 Estructura básica de los archivos Makefile

Cada archivo Makefile de la biblioteca tiene una estructura según la Figura 4.3

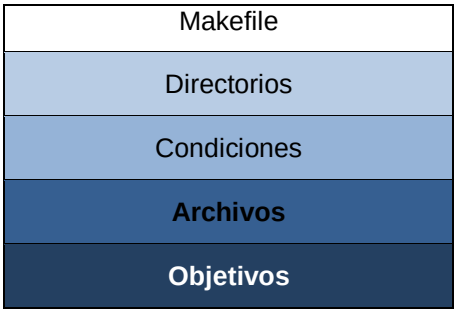

Figura 4.3 Estructura del archivo Makefile.

Directorios: Contiene asignaciones a todos los directorios utilizados por el archivo *Makefile*. El nombre de todas las asignaciones es igual en todos los *Makefile* como lo describe la Tabla 4.3.

Tabla 4.3 Asignación de directorios en el Makefile.

Parámetro	Ruta
BINDIR	$PCUDIR/lib
DOCDIR	$PCUDIR/doc
INCDIR	$PCUDIR/include
LIBDIR	$PCUDIR/lib
DSPDIR	$PCUDIR/lib/dsplib
SRCDIR	$PCUDIR/src

Condiciones: establece condiciones de ejecución del Makefile según los parámetros dados al ser llamada la aplicación *make*

Archivos: Es una asignación de todos archivos fuentes que necesita el Makefile para la ejecución.

Objetivos: Representa la parte principal del Makefile, cada objetivo representa una o varias instrucciones que deben de realizar para lograr el objetivo.

Capitulo 5. Bibliotecas PCU

5.1 Creación de una biblioteca

PCU es un conjunto de bibliotecas independientes interrelacionadas mutuamente según un nivel jerárquico. Esto permite al usuario determinar a qué nivel se quiere utilizar la biblioteca según la complejidad de la aplicación.

Para la compilación del código se utilizan las herramientas del *Toolchain*, compilador *arm-linux-gnueabi-gcc* (gcc) y archivador *arm-linux-gnueabi-ar* (ar).

Cada biblioteca fue creada utilizando el formato de biblioteca estática. Todas las bibliotecas se encuentran en el directorio $PCUDIR/lib y utilizan en el nombre, como estándar el sufijo ".a" para su identificación.

Para la realización de una biblioteca es necesario realizar los procesos de compilación y de enlazado en forma independiente. La Figura 5.1 muestra el proceso de creación de una biblioteca. Cada biblioteca se crea a partir de un archivo fuente y de uno o varios archivos cabecera. Cada archivo es compilado utilizando *gcc*.

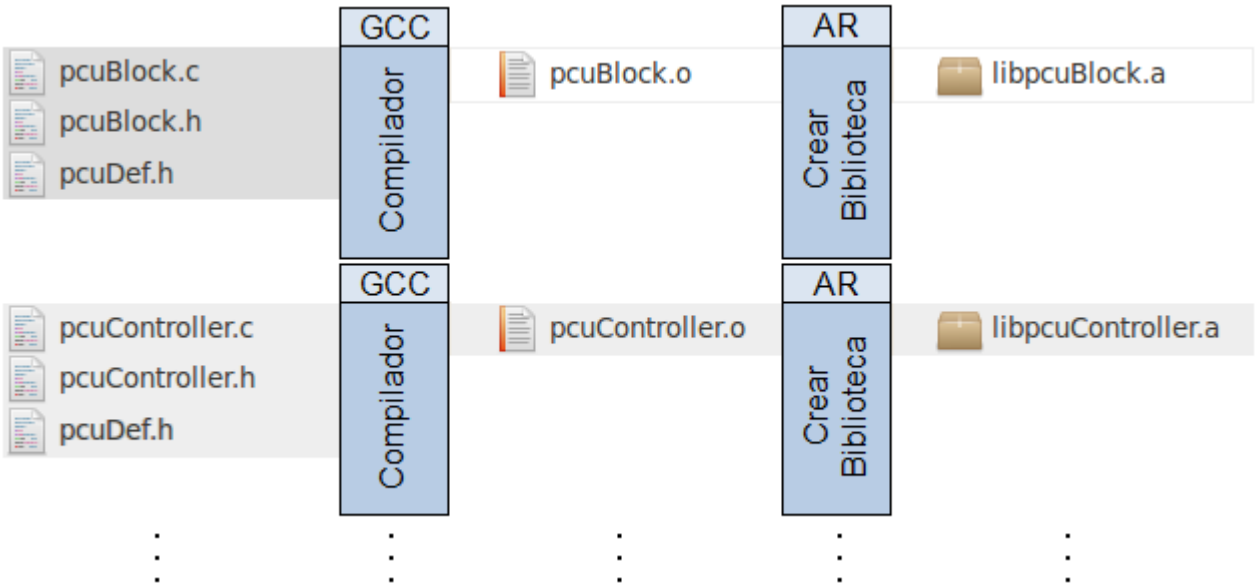

Figura 5.1 Proceso de creación de una biblioteca.

El compilador produce un archivo objeto con el código de la biblioteca y con llamadas a otras funciones, las cuales son referenciadas por medio de las funciones prototipo ubicadas en los archivos de cabecera. Utilizando el archivo objeto como entrada, la herramienta *ar,* (crea, modifica y extrae de archivos, utilizado para crear bibliotecas) crea un índice con los símbolos definidos en los archivos objeto. Una vez creado, este índice se actualiza en el archivo cada vez que *ar* realiza un cambio. Un archivo con este índice acelera la vinculación de la biblioteca, y permite a las rutinas de la biblioteca llamar a los demás rutinas sin tener en cuenta su ubicación en el archivo. La aplicación de las herramientas *gcc* y *ar* se repite para la creación de cada una de las bibliotecas.

Una vez compiladas las bibliotecas, éstas pueden ser utilizadas por el usuario para crear su aplicación específica como se muestra en la Figura 5.2, Este proceso se realiza en dos etapas: compilación y enlace.

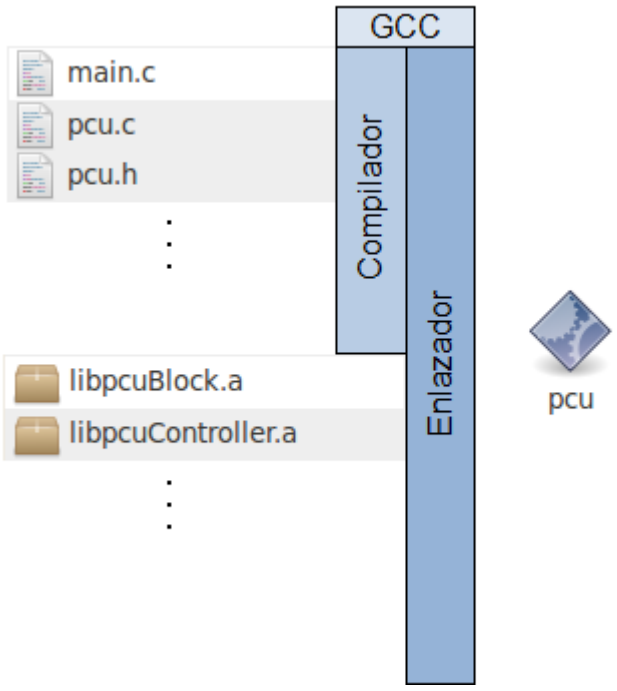

Figura 5.2 Proceso de creación de una aplicación.

El usuario crea sus propios archivos de código fuente los cuales contienen sus propias funciones y un punto de entrada a la aplicación (`main`) y archivos cabecera según sea necesario. El código fuente es compilado, una vez compilado se necesita incorporar el código binario de estas funciones a la aplicación que se está desarrollando por medio del enlazador, en esta etapa se reúne uno o más o módulos de código objeto con el código existente en las bibliotecas. El resultado es un archivo que contiene un programa autónomo, portable en cada sistema ya que el código binario se encuentra en el interior del ejecutable de la aplicación.

5.2 Creación de una biblioteca para el DSP

Esta es una biblioteca especial para la aplicación de algoritmos en el DSP. Esta biblioteca se diferencia de las demás bibliotecas que contienen módulos, que no ofrece funciones de creación, inicialización o destrucción de módulos, solo contiene funciones para la aplicación de algoritmos.

Esta biblioteca utiliza para su construcción el *toolchain* C6Run especial para desarrollos en el DSP. Las herramientas utilizadas son *c6runlib-cc* (*cc*) compilador del DSP para la creación de código objeto y *c6runlib-ar* (*ar*) para la creación de bibliotecas para el DSP.

Los pasos realizados para la creación de la biblioteca son iguales a una biblioteca para el GPP, solo cambia las herramientas utilizadas.

Paralelamente a esta biblioteca se crea una equivalente para el GPP, esta se crea con el objetivo de utilizar la biblioteca PCU eventualmente en plataformas que no contengan DSP, como principio para darle características de multiplataforma a la biblioteca PCU. Esta característica permite también evaluar rendimiento entre el uso de ambas bibliotecas.

La creación de ambas bibliotecas se realiza utilizando el mismo código fuente como se observa en la Figura 5.3.

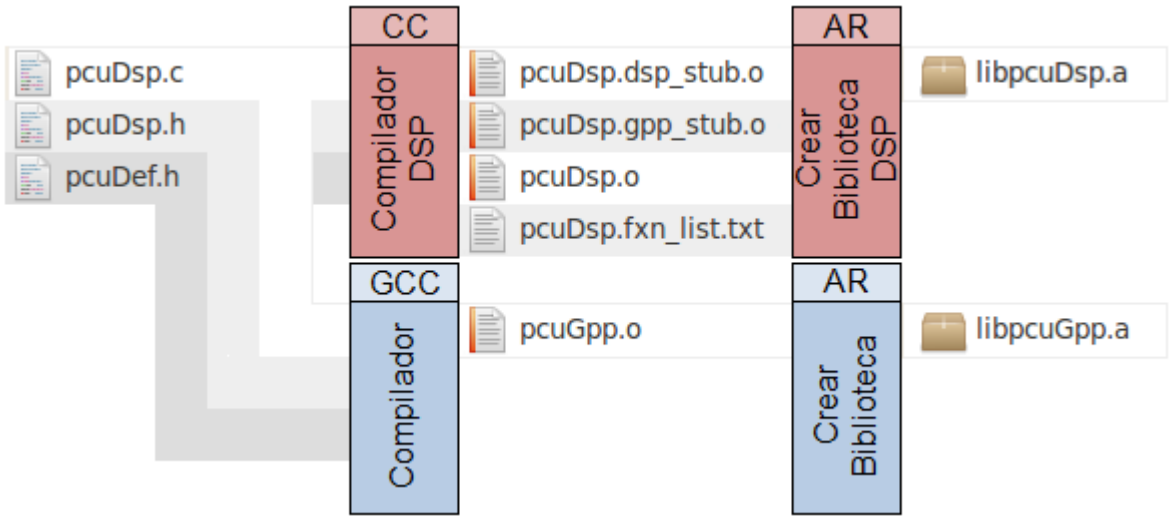

Figura 5.3 Proceso de creación de una biblioteca para el DSP.

El archivo fuente (pcuDsp.c) es compilado por c6runlib-cc produciendo el archivo objeto pcuDsp.o, además se produce dos archivos de código objeto adicionalmente.

Estos archivos son llamados *stubs* y son llamadas a procedimientos remotos, una para el GPP y otro para el DSP. Estos archivos objeto tienen el mismo nombre que el archivo de salida, pero con sufijo Gpp_stub.o y .Dsp_stub.o respectivamente para cada procesador. Un archivo con el sufijo .fxn_list.txt es generado y contiene la lista de funciones críticas que son ejecutadas en el DSP. Los archivos de salida de la herramienta *c6runlib-cc* tiene como destino la entrada de la herramienta archivador *c6runlib-ar* produciendo el archivo biblioteca libpcuDsp.a.

La herramienta c6runlib-ar permite que la etapa de enlace de la biblioteca con la aplicación sea transparente para el usuario y le proporciona acceso al DSP a través de las funciones de la biblioteca. Cuando la aplicación se ejecuta, el DSP se inicia y se carga con las funciones críticas para el DSP y espera las llamadas de las funciones desde el GPP, el cual a través de los *stubs* hace llamadas a las funciones remotas en el DSP. Véase Figura 5.4.

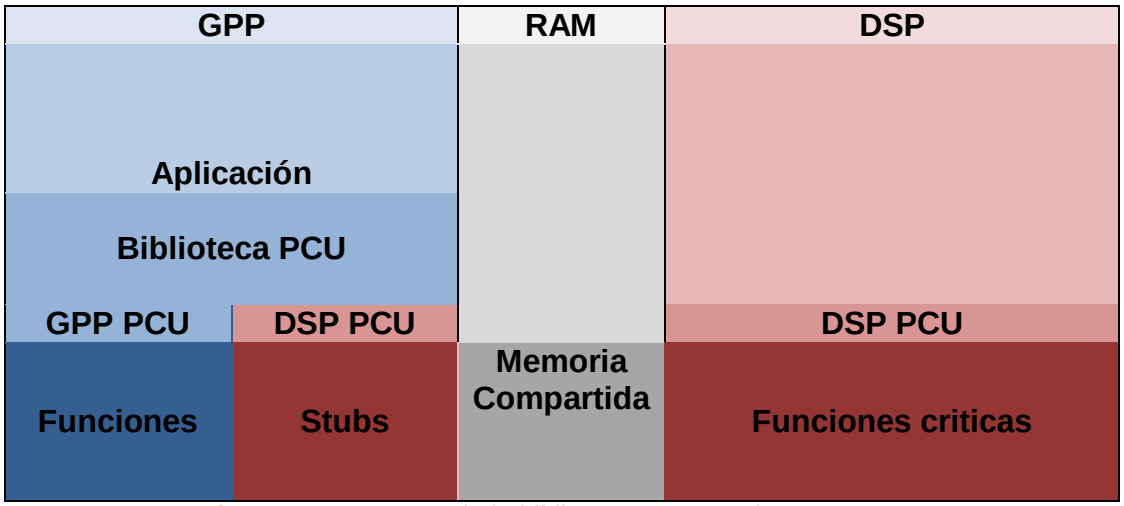

Figura 5.4 Esquema de la biblioteca PCU en el OMAP-L138.

La comunicación desde el GPP hacia el DSP se realiza por medio de búferes que son asignados como bloques de memoria contigua en RAM. El mecanismo para hacerlo es utilizando un administrador de memoria contigua llamado CMEM.

5.3 Organización lógica de la biblioteca PCU

La biblioteca PCU está compuesta por sub-bibliotecas las cuales contienen funciones específicas para cada uno de los niveles de abstracción.

Las bibliotecas tienen un orden jerárquico como lo muestra la Figura 5.5

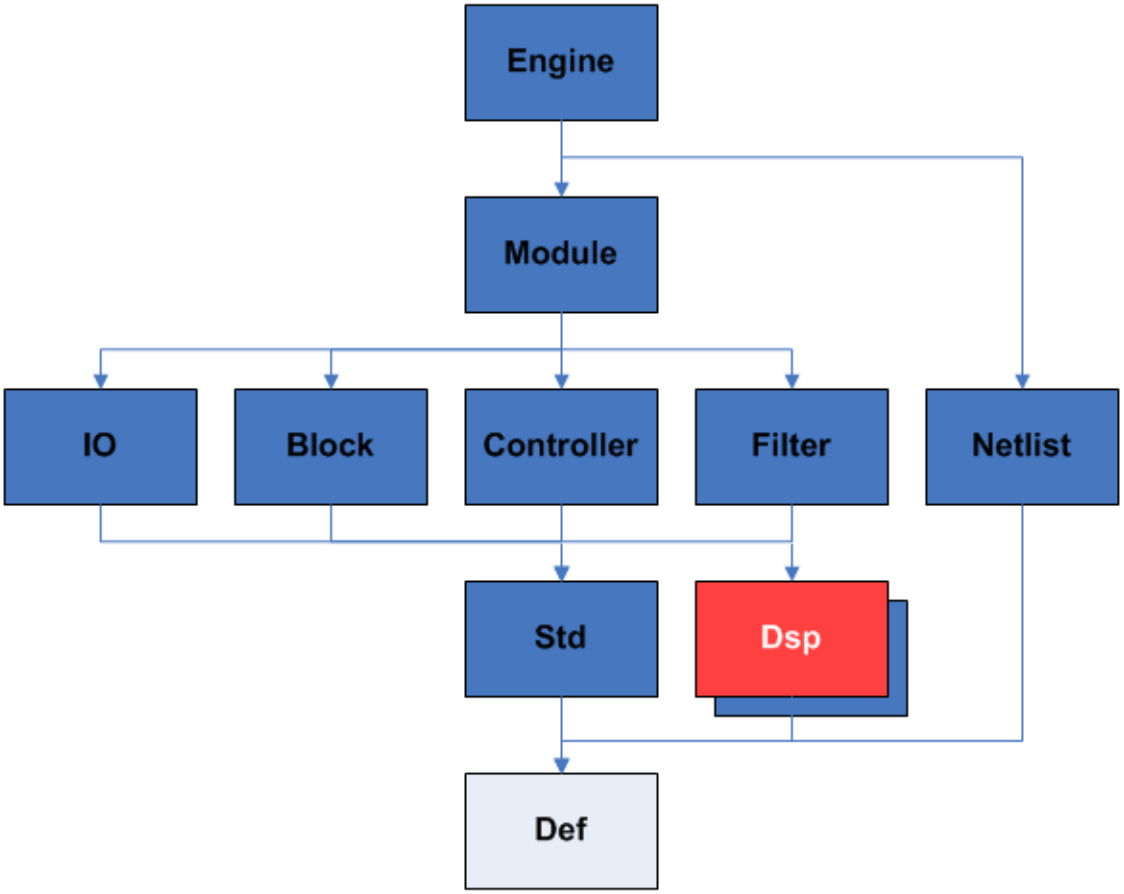

Figura 5.5 Jerarquía de la Biblioteca.

Cada biblioteca se describe con detalle en las siguientes secciones de este documento.

5.4 Descripción de las bibliotecas

5.4.1 Def

Este es un archivo cabecera (*pcuDef.h*) utilizado para definir todos los tipos de datos para la biblioteca. La definición de los tipos de datos propios para la biblioteca es necesaria, para que eventualmente pueda ser utilizada en otra plataforma con un tamaño de datos diferente. En este caso solo es necesario redefinir los tipos de datos para la plataforma deseada. La Tabla 5.1 presenta los tipos de datos definidos para la plataforma Hawkboard.

Tabla 5.1 Tipos de datos definidos en la biblioteca PCU.

Tipo de dato	Definición
PCUBYTE	Dato de 8 bits
PCUSBYTE	Dato de 8 bits con signo
PCUINT	Dato de 32 bits entero
PCUFP	Dato de 32 bits en punto flotante (utilizado en el GPP)
PCUFD	Dato de 32 bits en punto flotante (utilizado en el DSP)

El archivo también contiene todas definiciones usadas por la biblioteca en general, tales como el número de identificación de cada uno de los módulos soportados o de valores estándar de retorno de funciones mostrados en la Tabla 5.2.

Tabla 5.2 Valores estándar definidos en la biblioteca PCU

Etiqueta	Definición
PCU_SUCCESS	Valor de retorno de una función que indica que la función se realizó de forma satisfactoria
PCU_ERROR	Valor de retorno de una función que indica un error interno al realizar la función.
PCU_ERROR_MALLOC	Valor de retorno de una función que indica un error interno debido a la no asignación de memoria dinámica.
PCU_NULL	Variable para la asignación o comprobación de una variable nula

5.4.2 Std

Esta es una biblioteca estándar, contiene algoritmos para la conversión de datos. Por ejemplo, para la conversión de una cadena ASCII que representa un número Hexadecimal a binario. En esta biblioteca también se incluyen bibliotecas estándar del lenguaje C.

5.4.3 *Netlist*

Esta biblioteca permite manipular un archivo *netlist* para poder ser aplicado, (utilizado por la biblioteca *Engine*.)

Cada archivo *netlist* debe de tener la estructura de la Figura 5.6:

Descripción	Módulo fuente	Número	Salida	Módulo destino	Número	Entrada
Tamaño	Byte	Byte	Byte	Byte	Byte	Byte

Figura 5.6 Estructura de un enlace en un netlist.

Módulo fuente y Módulo destino: es un número binario que identifica cada uno de los módulos soportados por la aplicación.

Número: indica el número de módulo, cuando exista más de un módulo del mismo tipo en el sistema, la variable número permite identificar un módulo particular.

Salida: indica el número de salida del módulo (eventualmente para módulos de más de una salida).

Entrada: indica el número de entrada del módulo, para módulos con más de una entrada (por ejemplo, módulo de suma).

El formato anterior describe el enlace de dos módulos. Para la descripción de un sistema se necesita múltiples enlaces descritos en el archivo *netlist*. Cada enlace se encontrará en forma consecutiva dentro del archivo y contendrá tantos enlaces como

sea necesario. Cada *netlist* se divide en forma lógica en filas donde cada enlace corresponderá a una fila.

Un archivo *netlist* tiene dos funciones en la aplicación.

1. Descripción de la conexión de cada módulo del sistema de control.
2. Determinar el orden de ejecución de cada uno de los módulos.

La función `pcuNetlistSort` permite ordenar un archivo *netlist*. Esta función ordena cada una de las filas (enlaces) del archivo de forma tal que represente el orden de ejecución del sistema.

La Figura 5.7 muestra un ejemplo de sistema de control y su respectivo *netlist* en la Tabla 5.3.

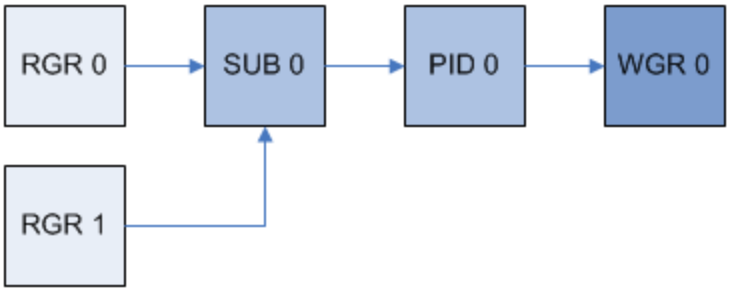

Figura 5.7 Sistema de control.

Tabla 5.3 Archivo Netlist.

```
PID 0x00 0x01 WGR 0x00 0x01
SUB 0x00 0x01 PID 0x00 0x01
RGR 0x00 0x01 SUB 0x00 0x01
RGR 0x01 0x01 SUB 0x00 0x02
```

El archivo debe contener al menos un módulo de entrada y un módulo de salida, características propias de un sistema de control.

La estructura del algoritmo de ordenamiento, (Figura 5.8) se basa en que cualquier módulo puede tener dos entradas, por lo tanto cada módulo es buscado dos veces como un módulo destino. Para realizar este proceso se utiliza una pila donde se añade cada módulo destino encontrado para luego volver a ser buscado posteriormente.

El uso de la pila permite al algoritmo desplazarse desde un módulo de salida hasta un módulo de entrada y luego devolverse por el mismo camino para buscar ramificaciones y ser procesadas.

El algoritmo inicia con un lazo que recorre todo el archivo, buscando un módulo de salida, si un módulo es encontrado éste se usa en un nuevo lazo, que recorre el archivo en busca del módulo como destino. Cuando se encuentra el módulo destino, se agrega a una pila y el módulo fuente es el siguiente destino a buscar. La fila del *netlist* es copiada a un nuevo *netlist* y se borra el módulo destino en el *netlist* original, esto permite que esta fila no vuelva a ser procesada.

Si no se encuentra un módulo destino (cuando se llega a los módulos de entrada ya que éstos nunca son módulos destino) el algoritmo toma el último módulo de la pila e inicia la búsqueda como módulo destino. Este proceso se repite hasta que la pila este vacía.

Si la pila está vacía, se inicia la búsqueda de un nuevo módulo de salida y se repite el proceso anterior.

Si no se encuentra un módulo salida, el algoritmo finaliza.

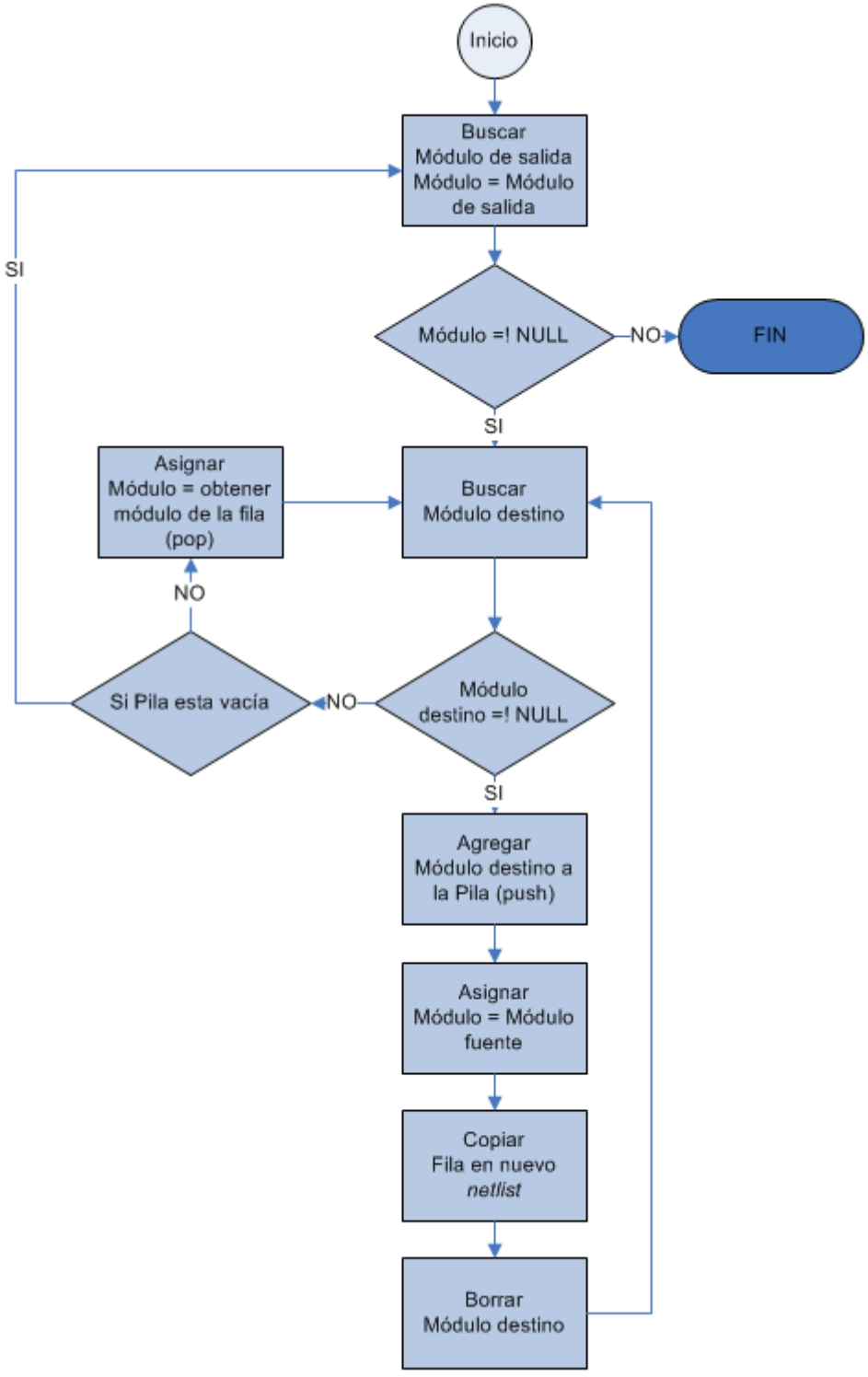

Figura 5.8 Algoritmo de ordenamiento.

Para el ejemplo de la Figura 5.7 el algoritmo genera como resultado el archivo mostrado en la Tabla 5.4.

Tabla 5.4 Archivo netlist generado por el algoritmo de ordenamiento.

RGR	0x00	0x01	SUB	0x00	0x01
RGR	0x01	0x01	SUB	0x00	0x02
SUB	0x00	0x01	PID	0x00	0x01
PID	0x00	0x01	WGR	0x00	0x01

Nótese que hacer un recorrido en forma de “L” en la tabla iniciando en la primera columna, permite obtener el orden de ejecución del sistema ejemplo, el cual es el siguiente: RGR 0, RGR 1, SUB 0, PID 0 y WGR 0.

5.4.4 Bibliotecas para el manejo de módulos

Este grupo de bibliotecas ofrecen al usuario una colección de módulos para la implementación de sistemas de control estáticos (no cambiantes en el tiempo) de forma directa. Un módulo es una estructura en memoria dinámica o estática según como sea creado por el usuario. Esta estructura es particular al tipo de módulo, y contiene las variables necesarias para su ejecución.

Cada módulo tiene una interfaz definida, mostrada en la Figura 5.9 para la comunicación externa con otro módulo. Esta interfaz está conformada por una o dos entradas y una salida.

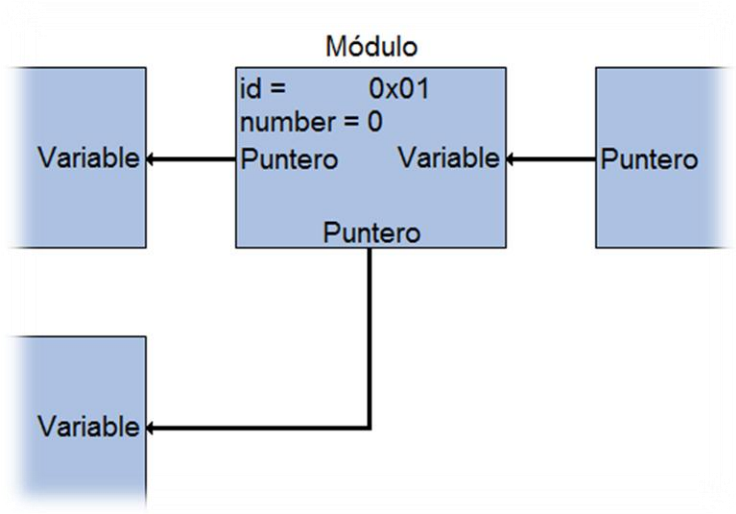

Figura 5.9 Conexión de módulos.

Las entradas son punteros, cada uno apunta hacia la salida del módulo anterior, la salida es una variable la cual cambia su contenido con la ejecución del módulo. La variable *id* contiene el número de identificación de cada módulo. Este número es único para cada módulo, y nunca debería ser cambiado por el usuario, la definición de este número se encuentra en el archivo *pcuDef.h*. La variable *number* es un número que permite identificar un módulo entre otros del mismo tipo.

Las bibliotecas para el manejo de módulos se dividen según su función como se observa en la Figura 5.10

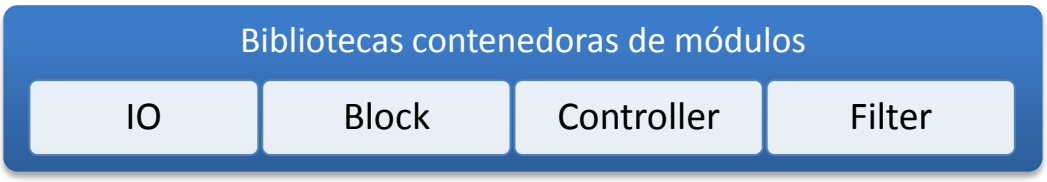

Figura 5.10 Bibliotecas contenedoras de módulos.

Cada biblioteca se describe a continuación:

a. *IO (Input/Output)* Esta biblioteca contiene todos los bloques de *hardware* para adquisición y transmisión de datos.
b. *Block*: Contiene todos los bloques funcionales en los cuales su aplicación se basa en una operación como resta o multiplicación.
c. *Controller*: Contiene controladores tales como PID.
d. *Filter*: biblioteca para la implementación de filtros digitales.

Cada módulo tiene una estructura propia la cual es creada, iniciada y destruida mediante funciones contenidas en cada biblioteca.

Cada módulo contiene las siguientes funciones.

a. `New`**:** Crea un módulo en memoria dinámica e inicia parcialmente las variables internas del módulo. (punteros de enlace, variables internas y registro de salida)
b. `Init`: Inicia el módulo, para los módulos que así lo requieran.
c. `Apply`: Ejecuta la función específica de cada módulo.
d. `Destroy`: Destruye el módulo creado.

Estas funciones son específicas para cada módulo. La utilización en la programación de cada una de las funciones indistintamente del módulo produce advertencias de incompatibilidad de tipo de datos en la compilación de la aplicación y errores en la ejecución.

La Figura 5.11 contiene el código para la ejecución del sistema de la Figura 5.7. Para la ejecución del sistema se declaran todos los módulos que lo conforman y se inicializa el módulo, de ser necesario. Se enlazan las salidas y entradas de cada módulo para crear el sistema. Una vez enlazados todos los módulos, el sistema puede ser procesado, ejecutando cada uno de los módulos secuencialmente, en el orden necesario para su correcto funcionamiento.

```
#include <pcuBlock.h>
#include <pcuController.h>
.
.
.
  PCURGR rgr0;                        // Crear RGR0
  PCURGR rgr1;                        // Crear RGR1
  PCUSUB sub0;                        // Crear SUB1
  PCUPID pid0;                        // Crear PID0
  PCUWGR wgr0;                        // Crear WGR0

  pcuRgrInit(&rgr0,0);                // Inicializar RGR0 r=0
  pcuRgrInit(&rgr1,-1.0);             // Inicializar RGR1 r=-1
  pcuPidInit(&pid0,1.0,2.0,3.0);      // Inicializar PID0 p=1 i=2 d=3
    // Enlazar módulos
  rgr0.input1 = &wgr0.output;
  rgr1.input1 = NULL;
  sub0.input1 = &rgr0.output;
  sub0.input2 = &rgr1.output;
  pid0.input1 = &sub0.output;
  wgr0.input1 = &pid0.output;
    // Ejecutar el sistema
  pcuRgrApply(&rgr0);
  pcuRgrApply(&rgr1);
  pcuSubApply(&sub0);
  pcuPidApply(&pid0);
  pcuWgrApply(&wgr0);
```

Creación	Inicialización	Enlace	Aplicación

Figura 5.11 Ejemplo de programación de un sistema utilizando bibliotecas contenedoras de módulos.

Debido a que cada uno de los módulos utiliza funciones específicas para cada una de las etapas, las bibliotecas no proveen funciones para el enlace entre dos módulos, su enlace se debe realizar con conocimiento de la estructura debido a que todas las funciones son únicas para cada módulo y no contienen funciones comunes para diferentes módulos.

5.4.5 *Module*

La biblioteca *Module* se creó para brindar al usuario un nivel superior de abstracción en la manipulación de cada uno de los módulos sin importar su tipo

particular. Un módulo es un componente del sistema, con una interfaz bien definida hacia otros componentes. Definir todos los tipos de bloques como módulos permite ver al sistema como un conjunto de piezas que se repiten en un sistema de cualquier tipo, para hacer más fácil y estándar su construcción. Esta característica permite la interrelación de módulos y crear sistemas de control de forma dinámica. La biblioteca permite realizar todas las funciones específicas de cada módulo tales como la creación, inicialización, ejecución y destrucción de forma general. Aunque la biblioteca permite la aplicación de cualquier módulo, esta biblioteca no está optimizada para la ejecución de un sistema. La biblioteca *Engine* contiene funciones para la ejecución de sistemas (ver sección *Engine*).

Cuando se crea un módulo es necesario manipularlo para enlazarlo con los demás módulos que conforman el sistema. Debido a que cada estructura particular de un módulo es diferente, la biblioteca *Module* posee un tipo de dato llamado PCUMOD (Figura 5.12) para el manejo general de módulos, descrito en la Tabla 5.5.

```
typedef struct {
  PCUBYTE id;
  PCUINT  number;
  void*   next;
  PCUFP   output;
  PCUFP*  input1;
  PCUFP*  input2;
}PCUMOD;
```

Figura 5.12 Tipo de dato Module (PCUMOD).

Para cada módulo particular se estableció una cabecera de datos estándar sin importar la cantidad de variables extras que debe contener la estructura para ejecutar su función particular. Esta cabecera debe ser conforme a la estructura PCUMOD. Esto permite que cada estructura particular pueda ser apuntada como un tipo de dato PCUMOD y permitir ser manipulada de forma general.

Tabla 5.5 Variables que conforma la estructura PCUMOD

Tipo de dato	Nombre	Definición
PCUBYTE	`id`	Variable que identifica el tipo de módulo a la cual pertenece la estructura.
PCUINT	`number`	Número de módulo.
void*	`next`	Puntero hacia un módulo.
PCUFP	`output`	Variable de salida.
PCUFP*	`input1`	Puntero de entrada.
PCUFP*	`input2`	Puntero de entrada.

Debido a que esta biblioteca se basa en este ordenamiento para realizar la manipulación de cada módulo, los nuevos módulos deben de contener esta cabecera en el orden y con los tipos de datos establecidos anteriormente, un cambio en la cabecera provoca errores en la manipulación y de segmentación por parte de esta biblioteca.

La Figura 3.1 es un ejemplo del uso de la cabecera en la estructura de programación de un controlador PID. Nótese que la cabecera contiene los tipos de datos y el mismo orden que la estructura PCUMOD.

```
typedef struct {
  PCUBYTE id;
  PCUINT number;
  void*  next;
  PCUFP  output;
  PCUFP* input;
  PCUFP  p;
  PCUFP  d;
  PCUFP  i;
  PCUFP  integ;
} PCUPID;
```

Variables de cabecera | Variables del Controlador

Figura 5.13 Estructura de programación de un Controlador PID.

La Figura 5.14 contiene el código para la ejecución del sistema de la Figura 5.7. La creación del sistema sigue la estructura utilizada por las bibliotecas contenedoras de módulos, creación de los módulos que lo conforma, inicialización del módulo,

enlace de las salidas y entradas de cada módulo para crear el sistema y aplicación de cada uno de los módulos secuencialmente en el orden correcto.

```
#include <pcuModule.h>
.
.
.
  PCUMH rgr0 = pcuModuleCreate(PCURGR_ID,0);     // Crear RGR0
  PCUMH rgr1 = pcuModuleCreate(PCURGR_ID,1);     // Crear RGR1
  PCUMH sub0 = pcuModuleCreate(PCUSUB_ID,0);     // Crear SUB0
  PCUMH pid0 = pcuModuleCreate(PCUPID_ID,0);     // Crear PID0
  PCUMH wgr0 = pcuModuleCreate(PCUWGR_ID,0);     // Crear WGR0

  pcuModuleInit(rgr0,0.0);          // Inicializar RGR0 r=0
  pcuModuleInit(rgr1,-1.0);         // Inicializar RGR1 r=-1
  pcuModuleInit(pid0,1.0,2.0,3.0); // Inicializar PID0 p=1 i=2 d=3
  // Enlazar módulos
  *(pcuModuleGetIn(rgr0,PCU_INPUT1)) = pcuModuleGetOut(wgr0,PCU_OUTPUT1);
  *(pcuModuleGetIn(rgr1,PCU_INPUT1)) = NULL;
  *(pcuModuleGetIn(sub0,PCU_INPUT1)) = pcuModuleGetOut(rgr0,PCU_OUTPUT1);
  *(pcuModuleGetIn(sub0,PCU_INPUT2)) = pcuModuleGetOut(rgr1,PCU_OUTPUT1);
  *(pcuModuleGetIn(pid0,PCU_INPUT1)) = pcuModuleGetOut(sub0,PCU_OUTPUT1);
  *(pcuModuleGetIn(wgr0,PCU_INPUT1)) = pcuModuleGetOut(pid0,PCU_OUTPUT1);
  // Ejecutar el sistema
  pcuModuleApply(rgr0);
  pcuModuleApply(rgr1);
  pcuModuleApply(sub0);
  pcuModuleApply(pid0);
  pcuModuleApply(wgr0);
  // Liberar memoria reservada
  pcuModuleDestroy(rgr0);
  pcuModuleDestroy(rgr1);
  pcuModuleDestroy(sub0);
  pcuModuleDestroy(pid0);
  pcuModuleDestroy(wgr0);
```

Creación	Inicialización	Enlace	Aplicación

Figura 5.14 Ejemplo de programación de un sistema utilizando la biblioteca Module.

Nótese que en cada una de las etapas se utiliza la misma función para ejecutar cada acción, la diferenciación entre cada uno de los módulos se realiza en la etapa de creación, utilizando como argumentos el identificador de cada módulo definido en la biblioteca. La biblioteca fue construida para su implementación de forma dinámica por lo tanto cada módulo creado utiliza una asignación de memoria en el sistema la cual debe ser liberada al final de la ejecución por medio de las funciones `Destroy`.

5.4.6 *Engine*

Engine es el núcleo de la biblioteca PCU. Permite ejecutar cada uno de los módulos que conforman un proceso. *Engine* es una estructura que permite realizar las siguientes funciones:

a. Permite guardar todos los módulos creados y poder ser accedidos posteriormente.
b. Permite establecer el orden de ejecución de un proceso y lo mantiene dentro de la estructura.
c. Permite ejecutar un proceso del sistema, descrito a través del archivo *netlist* dado por el usuario y procesado por la biblioteca *Netlist* en el orden de ejecución establecido.
d. Elimina todos los módulos guardados.

Cada estructura de datos *Engine* permite contener infinita cantidad de módulos (la cantidad es limitada por la memoria disponible en sistema operativo). La utilización de un módulo es definida por el archivo *netlist.* Si un módulo no es utilizado se mantendrá dentro de la estructura hasta que ésta sea destruida. Lo anterior permite, ejecutar uno o más procesos, definiendo diferentes procesos por medio de archivos netlist. La biblioteca contiene funciones para la creación inicialización, aplicación y destrucción de la estructura *Engine* mostrada en la Figura 5.15.

```
typedef struct {
  PCUBYTE   number;
  PCUINT    mod_num;
  PCUINT    exe_num;
  PCUMH*    exe_mod;
  PCUMOD*   mod[PCUMOD_NUM];
  PCUMODULEAPPLY EngApply[PCUMOD_NUM];
}PCUENG;
```

Figura 5.15 Estructura de programación de Engine.

Cada una de las variables es descrita en la Tabla 5.6.

Tabla 5.6 Variables que contiene la estructura Engine.

Variable	Descripción
`number`	Es número que permite identificar entre dos estructuras *Engine*.
`mod_num`	Cantidad de módulos contenidos por la estructura *Engine*.
`exe_num`	Cantidad de módulos a ser ejecutados
`exe_mod`	Puntero a una lista de los módulos a ejecutar.
`mod`	Estructura para guardar a todos los módulos creados.
`EngApply`	Arreglo de funciones de aplicación para cada uno de los módulos.

La lista hacia la que apunta `exe_mod` contiene los módulos en el orden de ejecución necesario para realizar el proceso. Esta lista es generada por medio del *netlist* dado por el usuario (el orden de ejecución es establecido por la biblioteca *Netlist*).

La lista de ejecución consiste en una lista de punteros a módulos. Esta lista se recorre consecutivamente para la ejecución de todo el sistema y su longitud es establecida en la variable `exe_num`. Al hacer un llamado a la función `pcuEngineApply` el primer módulo de la lista será el primero en ejecutarse y seguirá ejecutando módulos consecutivamente hasta llegar al último módulo. Todos los módulos que se encuentren en la lista deben haber sido agregados previamente a la estructura *Engine* para realizar su ejecución.

Todos los módulos agregados a *Engine* son guardados por medio de la estructura `mod`. Esta estructura es mostrada con más detalle en a Figura 5.16.

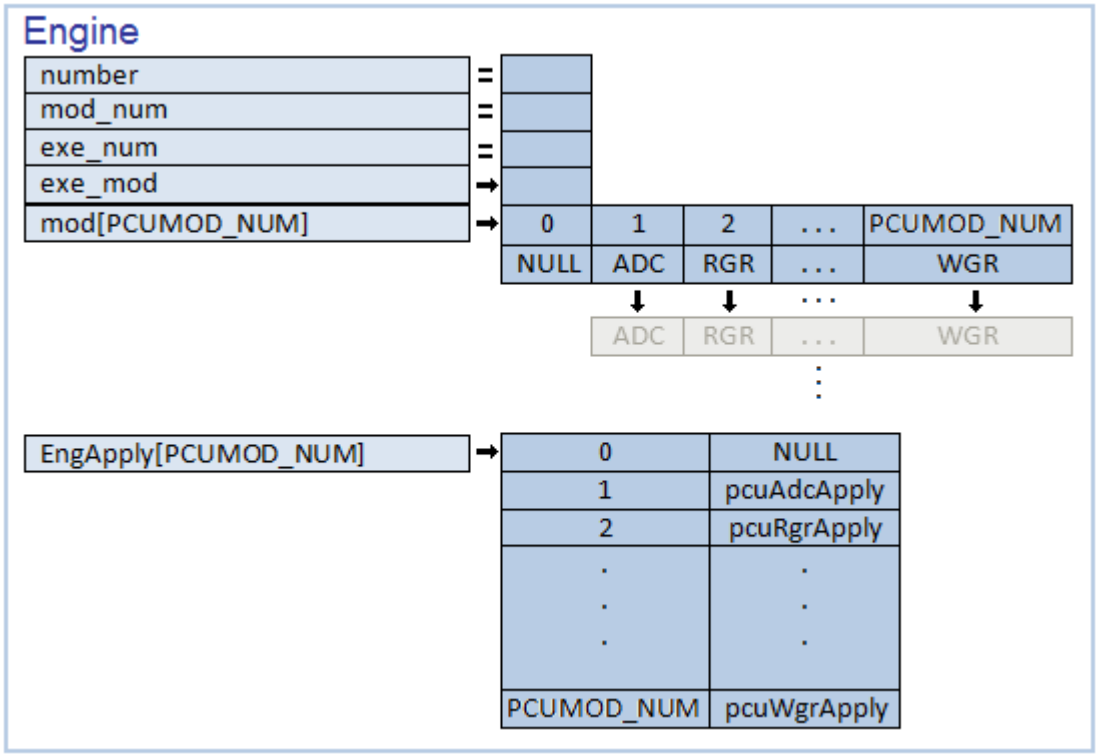

Figura 5.16 Diagrama gráfico de la estructura Engine.

La variable `mod` es un arreglo de punteros tipo PCUMOD (Figura 5.12) de tamaño igual al número de módulos soportados por el sistema. Cada tipo de módulo tiene una ubicación fija dentro del arreglo. Cuando se agrega un módulo dentro de la estructura, el módulo es apuntado por una posición del arreglo, y así con cada uno de los módulos agregados. Cuando se agregan dos o más módulos del mismo tipo, el primer módulo es apuntado por una posición del arreglo y el siguiente módulo es apuntado por la variable `next` del módulo agregado anteriormente. Este tipo de enlace se repite para cada nuevo módulo agregado del mismo tipo, formando así una lista enlazada de módulos.

La variable `pcuModuleApply` es un arreglo de punteros a funciones. Este arreglo apunta hacia todas las funciones tipo *Apply* de los módulos soportados por la biblioteca. Esta estructura es utilizada para ejecutar cada módulo.

La Figura 5.18 contiene el código para la ejecución del sistema de la Figura 5.7 de forma dinámica, utilizando la estructura *Engine,* la cual es creada e inicializada cada una de sus variables internas y apuntadas cada una de las funciones *Apply* de cada módulo. Para la ejecución del sistema se declaran dos arreglos para simular archivos: `system` el cual contiene los módulos, el número de cada módulo y su respectiva inicialización en el orden establecido en la Figura 5.17 y la estructura `netlist` la cual se describe en la Figura 5.6.

Descripción	Módulo ID	Número	Valores de inicialización		
			Argumento 1	Argumento 2	Argumento 3
Tamaño	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes

Figura 5.17 Estructura del arreglo system.

La estructura `netlist` es recorrida por medio de un lazo que asigna de forma dinámica el valor de “Módulo ID” y “número” a la función de creación de módulos realizando un coercionamiento de la variable, y los valores de inicialización, a la respectiva función de inicialización de módulos. Posteriormente se agrega el módulo a la estructura *Engine.*

La función `pcuEngineSetExeMod` establece el orden de ejecución del sistema y enlaza las salidas y las entradas de cada módulo. La aplicación de la estructura *Engine* ejecuta internamente cada módulo según la estructura `netlist`.

```
#include <pcuEngine.h>
.
.
.
PCUFP system[] = {
    PCURGR_ID,0, 0.0,  0,  0,
    PCURGR_ID,1,-1.0,  0,  0,
    PCUSUB_ID,0,   0,  0,  0,
    PCUPID_ID,0, 1.0,2.0,3.0,
    PCUWGR_ID,0,   0,  0,  0
  };
  PCUBYTE netlist[] = {
    PCUPID_ID,0,PCU_OUTPUT1,PCUWGR_ID,0,PCU_INPUT1
    PCUSUB_ID,0,PCU_OUTPUT1,PCUPID_ID,0,PCU_INPUT1,
    PCURGR_ID,0,PCU_OUTPUT1,PCUSUB_ID,0,PCU_INPUT1,
    PCURGR_ID,1,PCU_OUTPUT1,PCUSUB_ID,0,PCU_INPUT2,
  };
  PCUINT i;
  PCUMH mod;                              // Manejador de un módulo (Handle)
  PCUEH Engine = pcuEngineNew(0);         // Crear Engine
  pcuEngineInit(Engine);                  // Inicializar Engine

  for(i=0;i<sizeof(system)/sizeof(PCUFP);i+=5){
    mod = pcuModuleCreate((PCUINT)system[i],       // Crear módulo
                          (PCUINT)system[i+1]);
    pcuModuleInit(mod,system[i+2],system[i+3],     // Inicializar módulo
                                  system[i+4]);
    pcuEngineAddMod(Engine,mod);                 // Agregar módulo en Engine
  }
  // Establecer la ejecución y enlazar los módulos
  pcuEngineSetExeMod(Engine,sizeof(netlist),&netlist[0]);
  // Ejecutar el sistema
  pcuEngineApply(Engine);                  // Ejecutar el sistema

  pcuEngineDestroy(Engine);
```

Creación	Inicialización	Enlace	Aplicación

Figura 5.18 Ejemplo de programación para la creación dinámica de un sistema utilizando la biblioteca Engine.

La función `pcuEngineDestroy` libera la memoria reservada por la estructura *Engine* asi como de todos módulos guardados dentro de la estructura.

Capitulo 6. Resultados

La biblioteca PCU tiene diferentes niveles de abstracción según la biblioteca que se utilice. Cada nivel de abstracción puede implicar un aumento de procesamiento lo cual se refleja en el tiempo de ejecución. La Tabla 3.1 muestra los tiempos obtenidos en la ejecución del un módulo PID utilizando diferentes bibliotecas.

Tabla 6.1 Tiempo de ejecución de un módulo PID según la biblioteca utilizada.

Controller		*Module*		*Engine*	
Tiempo de Ejecución (µs)	Tiempo promedio (µs)	Tiempo de Ejecución (µs)	Tiempo promedio (µs)	Tiempo de Ejecución (µs)	Tiempo promedio (µs)
3,5	3,7 σ=0,63	6,3	5,3 σ=0,82	8,9	6,3 σ=1,07
4,5		6,3		5,9	
3,5		5,3		5,9	
4,5		5,3		5,9	
3,5		5,3		5,9	
3,5		5,3		6,9	
4,5		4,3		6,9	
3,5		4,3		5,9	
3,5		4,3		4,9	
2,5		6,3		5,9	

La Figura 6.1 muestra gráficamente el tiempo promedio utilizado por cada biblioteca para la ejecución del controlador PID en el GPP (ARM9). La implementación directa del PID por medio de la biblioteca *Controller* presenta el menor tiempo de ejecución, un mayor nivel de abstracción utilizando la biblioteca *Module* conlleva un aumento 44% en el tiempo de procesamiento con respecto a la implementación directa, a su vez la implementación por medio de *Engine* conlleva un aumento de 19% con respecto a la ejecución utilizando Module y un 70% con respecto a la implementación directa.

Cuando la aplicación es estática, es decir, la estructura de control siempre es la misma después de construida y no cambia en el tiempo, la implementación directa

representa la forma óptima de implementación, utilizándose el menor tiempo de procesamiento.

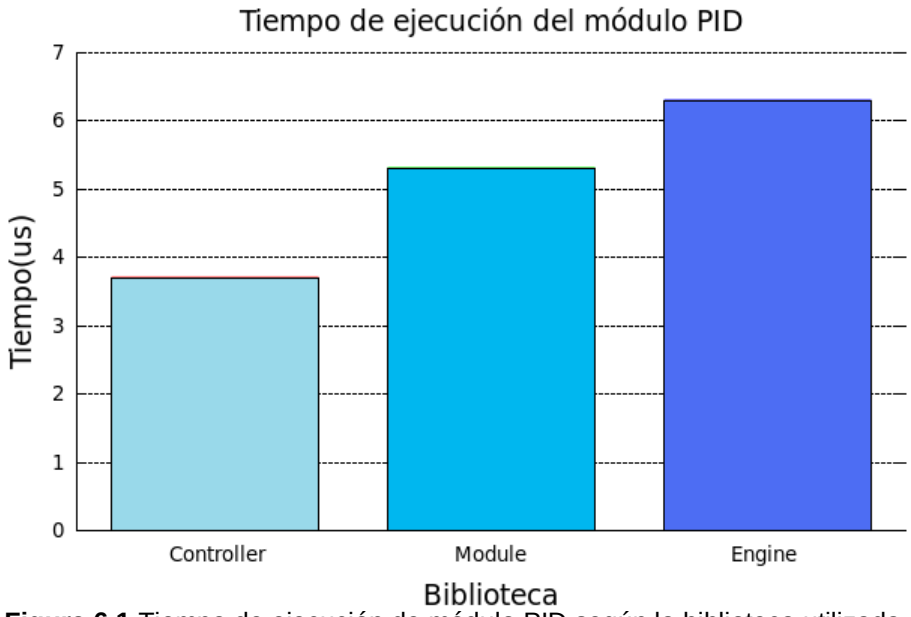

Figura 6.1 Tiempo de ejecución de módulo PID según la biblioteca utilizada.

El aumento en el procesamiento que se obtiene al utilizar las bibliotecas *Module* y *Engine* se justifica cuando hay la necesidad de crear un sistema de control dinámico. Los resultados de la Tabla 6.2 muestran el desempeño de cada biblioteca para el sistema de la Figura 5.7.

Tabla 6.2 Tiempo de ejecución de un sistema según la biblioteca utilizada.

Controller		*Module*		*Engine*	
Tiempo de Ejecución (µs)	Tiempo promedio (µs)	Tiempo de Ejecución (µs)	Tiempo promedio (µs)	Tiempo de Ejecución (µs)	Tiempo promedio (µs)
8,0	6,7 σ=0,82	10,0	9,68 σ=0,69	8,0	7,1 σ=0,57
7,0		9,2		7,0	
6,0		10,2		7,0	
7,0		9,2		7,0	
8,0		10,2		7,0	
6,0		9,2		6,0	
6,0		9,2		7,0	
7,0		9,2		7,0	
6,0		11,2		8,0	
6,0		9,2		7,0	

La implementación directa siempre será la forma óptima de programación. Como se menciona anteriormente, el uso de las bibliotecas *Module* o *Engine* se justifican por el tipo de aplicación. La biblioteca *Module* fue creada para la manipulación de módulos, pero no para la ejecución de los mismos, por lo tanto su uso para la ejecución de módulos presenta el mayor tiempo de ejecución como lo muestra la Figura 6.2.

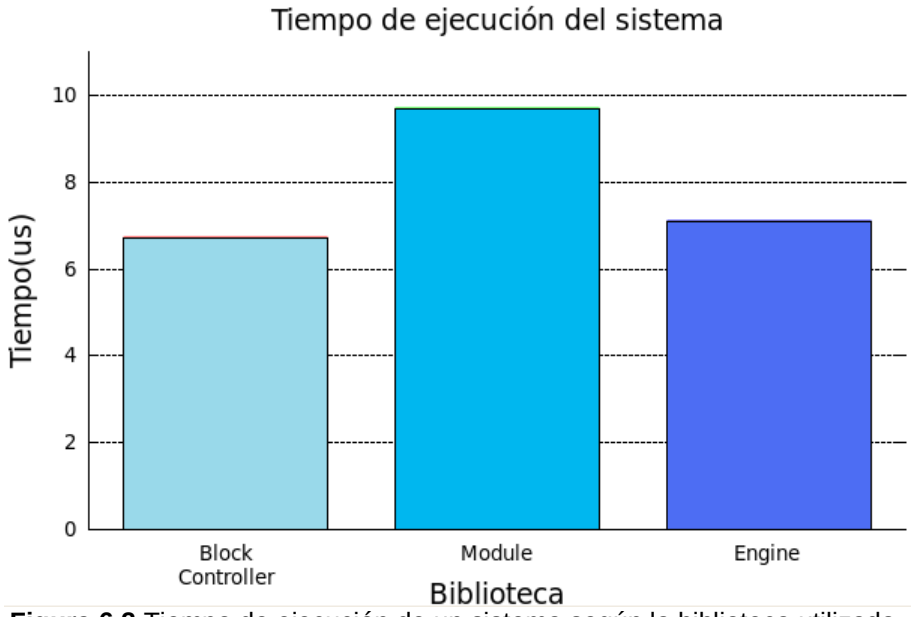

Figura 6.2 Tiempo de ejecución de un sistema según la biblioteca utilizada.

La biblioteca *Engine* esta optimizada para la ejecución dinámica de sistemas. La ejecución por medio de *Engine* necesita un 6% más de tiempo que la implementación directa para el sistema evaluado, lo cual demuestra la eficiencia de la biblioteca para la ejecución de múltiples módulos.

La biblioteca *Filter* permite al usuario la ejecución de módulos en el DSP, esta implementación permite liberar al GPP de una carga de procesamiento y pasársela al DSP para tareas especializadas. Para la evaluación se creó el sistema mínimo de la Figura 6.3.

Figura 6.3 Filtro FIR.

El sistema esta compuesto por un registro general de lectura (RGR), un filtro FIR el cual procesa un dato por cada ejecución y un registro general de escritura (WGR)

Se ejecutó en el sistema en el GPP y en el GPP+DSP para obtener los datos de la Tabla 6.3.

Tabla 6.3 Porcentaje de uso del CPU.

Tiempo(s)	GPP Uso del CPU (%)	GPP+DSP Uso del CPU (%)
0	88	70
1	97	78
2	96	75
3	95	67
4	96	76
5	97	69
6	98	76
7	97	76
8	96	65
9	90	71
10	88	75
11	97	74
12	95	82
13	96	74
14	97	81
15	88	69
16	97	71
17	95	71
18	96	73
19	97	66
20	88	66
21	97	74
22	95	72
23	96	66
24	98	68
25	95	68
26	96	71
27	97	64
28	98	70
29	97	71
30	96	71
	Promedio=95,13 σ=3,14	Promedio=71,61 σ=4,48

El uso del procesador se ve disminuido cuando se realiza una implementación DSP+GPP como se observa en la Figura 6.4. La aplicación solo ejecutada en el GPP tiene utiliza el CPU en promedio un 95% (el 5% restante es utilizado para ejecutar tareas propias del sistema operativo) mientras que la ejecución en el GPP+DSP utiliza el CPU en promedio un 72% liberando de procesamiento al GPP.

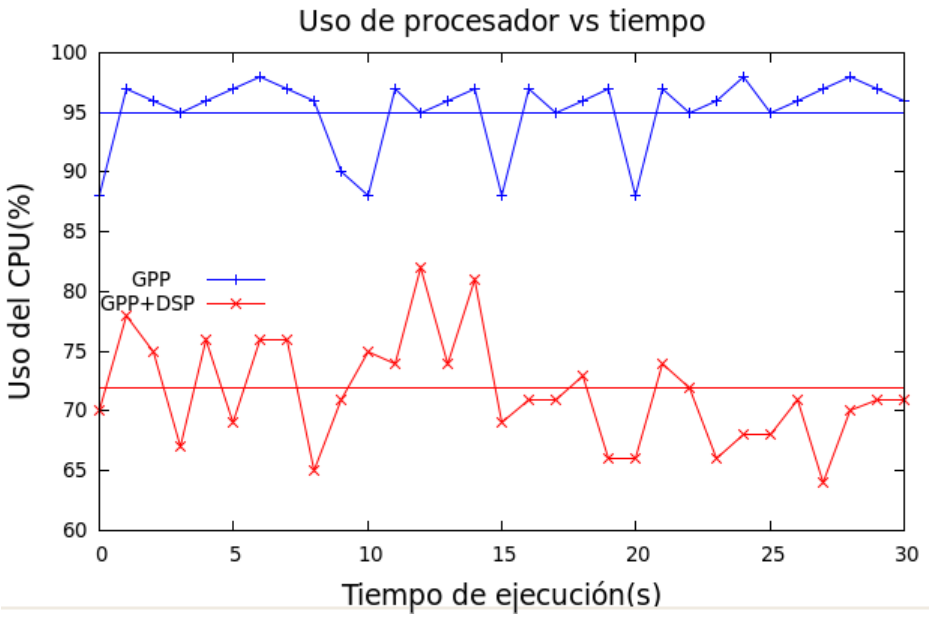

Figura 6.4 Uso del procesador vs tiempo de ejecución.

Capitulo 7. Conclusiones y Recomendaciones

7.1 Conclusiones

La utilización de diferentes niveles de abstracción de las bibliotecas para el manejo de módulos permite la optimización del código según la aplicación a desarrollar.

El desarrollo de una biblioteca para creación dinámica de sistemas permite crear una aplicación única por parte del usuario para la ejecución de múltiples sistemas, utilizando solamente parámetros de entrada a la aplicación tales como archivos de configuración.

El uso de la biblioteca permite que el usuario solo se deba de enfocarse en la aplicación y no en la creación de cada módulo de control, reduciendo el tiempo de desarrollo de la aplicación en la plataforma OMAP-L138.

La biblioteca permite la implementación de módulos en el DSP de forma directa y transparente para el usuario de la biblioteca.

La implementación de módulos en el DSP permite disminuir el porcentaje de uso del GPP para la ejecución de aplicaciones.

7.2 Recomendaciones

Permitir acceso libre al repositorio de la biblioteca para promover el uso de la misma y crear una retroalimentación para su optimización y desarrollo.

Crear una estructura de ejecución más compleja, que permitan el paralelismo del GPP con el DSP logrando un tiempo de ejecución menor de un sistema.

Desarrollar las bibliotecas para la entrada y salida de señales por medio de convertidores analógico a digital y digital a analógico, modulación por ancho de pulso

o periféricos de comunicación digital como SPI, I^2C, *Universal Parallel Port* así como el *hardware* necesario para su aplicación.

Investigar sobre la creación de una biblioteca para el uso de la *Programmable Real-Time Unit Subsystem* (PRUSS) de la plataforma OMAP-L138, el cual consta de dos núcleos RISC de 32 bits con controlador de interrupciones, memoria local de instrucciones y datos e interfaces de captura, con tiempos de ejecución deterministas.

Capitulo 8. Bibliografía

[1] Caravaca, Oscar. *Diseño de un Entorno de Desarrollo Integrado para una Unidad Controladora de Procesos. 2010*. [en línea] Agosto 2010. URL http://www.ie.itcr.ac.cr/einteriano/control/Laboratorio/Proyectos/2010_IDE/IDE_Informe_Final.pdf

[2] Castro, Daniel. *Unidad Controladora de Procesos para el diseño, análisis, simulación e implementación de sistemas de control automático a través de redes TCP/IP*, 2008. [en línea] Agosto 2010. URL http://www.ie.itcr.ac.cr/einteriano/control/Laboratorio/Proyectos/2008_BPC/BPC_InformeFinal.pdf

[3] Code Sourcery G++. *GNU Toolchain*. [en línea] Agosto 2010. URL http://www.codesourcery.com/sgpp/lite/arm

[4] Denx. Software Engineering. *Das U-Boot - the Universal Boot Loader*. [en línea] Agosto 2010. URL http://www.denx.de/wiki/U-Boot

[5] Doxygen Project. *Generate documentation from source code* [en línea] Agosto 2010. URL http://www.doxygen.org/

[6] GNU org. *make utility* [en línea] Agosto 2010. RL http://www.gnu.org/software/make/

[7] Hawkboard.org. *User's Manual*. Disponible en: URL http://www.hawkboard.org/

[8] Infocenter Arm. *ELF for the ARM Architecture*. Disponible en: URL http://infocenter.arm.com/help/topic/com.arm.doc.ihi0044d/IHI0044D_aaelf.pdf

[9] Katsuhiko, Ogata. *Sistemas de Control en Tiempo Discreto*. Segunda Edición. Prentice Hall Hispanoamericana, S.A, 1996.

[10] Leiva Manuel, *Process Controller Unit API*. Guia para desarrolladores. 2010

[11] Linux online. *Linux* [en línea] Agosto 2010. URL http://www.linux.org/

[12] Linux español. *Kernel* [en línea] Agosto 2010. URL http://www.linux-es.org/kernel

[13] Linux Documentation project. *Program Library*. [en línea] Agosto 2010 URL http://tldp.org/HOWTO/Program-Library-HOWTO/index.html

[14] RidgeRun Embedded Solution. *SDK User's Guide*

[15] Subversion Apache. *Subversion*. [en línea] Agosto 2010 URL http://subversion.apache.org/

[16] Texas Instruments. *CMEM* [en línea] Setiembre 2010. URL http://processors.wiki.ti.com/index.php/CMEM_Overview

[17] Texas Instruments*. Codec Engine Application Developer User's Guide. (SPRUE67D)* Disponible en: URL http://focus.ti.com/lit/ug/sprue67d/sprue67d.pdf

[18] Texas Instrument. *C6Run*. [en línea] Setiembre 2010.
URL http://processors.wiki.ti.com/index.php/C6Run

[19] Texas Instrument. *C6RunApp*. [en línea] Setiembre 2010.
URL http://processors.wiki.ti.com/index.php/C6RunApp_Documentation

[20] Texas Instrument. *C6RunLib*. [en línea] Setiembre 2010.
URL http://processors.wiki.ti.com/index.php/C6RunLib_Documentation

[21] Texas Instrument. *DSP/BIOS LINK* Version 1.65.00.02

[22] Texas Instrument. *OMAP-L138 Low-Power Applications Processor*. Disponible en: URL http://focus.ti.com/docs/prod/folders/print/omap-l138.html

[23] Tool Interface Standard. *Executable and Linking Format (ELF) Specification*. Disponible en: URL http://refspecs.freestandards.org/

[24] Wikimedia Foundation*. Application binary interface* [en línea] Setiembre 2010.
URL http://en.wikipedia.org/wiki/Application_binary_interface

[25] Wikimedia Foundation. *Bootloader* [en línea] Setiembre 2010.
URL http://es.wikipedia.org/wiki/Bootloader

Apéndice

A.1 Lista de archivos de la biblioteca PCU

Archivo	Directorio	Descripción
Makefile	$PCUDIR/	Archivo para la construcción general de la biblioteca PCU
Doxyfile	$PCUDIR/doc/	Archivo para la generación automática de documentación de código
ChangeLog	$PCUDIR/doc/ documentation/	Historial de cambios de la biblioteca PCU
pcu_api_guia_desarrollador.pdf	$PCUDIR/doc/ documentation/	Guia para el desarrollador de la biblioteca PCU
pcu_dsp_guia_desarrollador.pdf	$PCUDIR/doc/ documentation/	Guia para el desarrollador de la biblioteca para el DSP
sdk_guia_usuario.pdf	$PCUDIR/doc/ documentation/	Guia de usuario para la utilización del SDK de RidgeRun
hawkboard.png	$PCUDIR/doc/ images/	Archivo imagen utilizada para la documentación del código
pid.png	$PCUDIR/doc/ images/	Archivo imagen utilizada para la documentación del código
pcuBlock.h	$PCUDIR/include/	Archivo cabecera de la biblioteca *Block*
pcuController.h	$PCUDIR/include/	Archivo cabecera de la biblioteca *Controller*
pcuDef.h	$PCUDIR/include/	Archivo de definiciones de la biblioteca PCU
pcuDsp.h	$PCUDIR/include/	Archivo cabecera de la biblioteca *Dsp*
pcuEngine.h	$PCUDIR/include/	Archivo cabecera de la biblioteca *Engine*
pcuFilter.h	$PCUDIR/include/	Archivo cabecera de la biblioteca *Filter*
pcuIO.h	$PCUDIR/include/	Archivo cabecera de la biblioteca *Input/Output*
pcuModule.h	$PCUDIR/include/	Archivo cabecera de la biblioteca *Module*
pcuNetlist.h	$PCUDIR/include/	Archivo cabecera de la biblioteca *Netlist*
pcuStd.h	$PCUDIR/include/	Archivo cabecera de la biblioteca *Std*
Makefile	$PCUDIR/lib/	Archivo para la construcción automática de la biblioteca PCU
pcuBlock.c	$PCUDIR/lib/	Código fuente de la biblioteca *Block*
pcuController.c	$PCUDIR/lib/	Código fuente de la biblioteca *Controller*
pcuEngine.c	$PCUDIR/lib/	Código fuente de la biblioteca *Engine*
pcuFilter.c	$PCUDIR/lib/	Código fuente de la biblioteca *Filter*
pcuIO.c	$PCUDIR/lib/	Código fuente de la biblioteca *Input/Output*
pcuModule.c	$PCUDIR/lib/	Código fuente de la biblioteca *Module*
pcuNetlist.c	$PCUDIR/lib/	Código fuente de la biblioteca *Netlist*
pcuStd.c	$PCUDIR/lib/	Código fuente de la biblioteca *Std*
Makefile	$PCUDIR/lib/dsplib	Archivo para la construcción automática de la biblioteca para el DSP
pcuDsp.c	$PCUDIR/lib/dsplib	Código fuente de la biblioteca *Dsp*
main.c	$PCUDIR/lib/dsplib	Archivo para crear aplicaciones de prueba para el DSP
Makefile	$PCUDIR/src/	Archivo para la construcción automática de una aplicación utilizando la biblioteca PCU

Main	$PCUDIR/src/	Código fuente ejemplo de una aplicación utilizando la biblioteca PCU
pcu.c	$PCUDIR/src/	Código fuente de las funciones propias a la aplicación
pcu.h		Archivo cabecera del archivo pcu.c